

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日
Date of Application:

2002年 7月31日

出 願 番 号
Application Number:

特願2002-224207

[ST.10/C]:

[JP2002-224207]

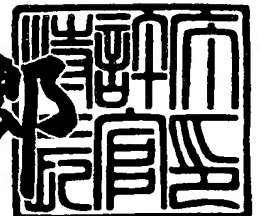
出 願 人
Applicant(s):

インターナショナル・ビジネス・マシーンズ・コーポレーション

2003年 2月 7日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3005353

【書類名】 特許願
【整理番号】 JP9010303
【提出日】 平成14年 7月31日
【あて先】 特許庁長官 殿
【国際特許分類】 G06F 9/45
【発明者】

【住所又は居所】 神奈川県大和市下鶴間 1 6 2 3 番地 1 4 日本アイ・ピー・エム株式会社 東京基礎研究所内

【氏名】 菅沼 俊夫

【発明者】

【住所又は居所】 神奈川県大和市下鶴間 1 6 2 3 番地 1 4 日本アイ・ピー・エム株式会社 東京基礎研究所内

【氏名】 小笠原 武史

【特許出願人】

【識別番号】 390009531

【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】 100086243

【弁理士】

【氏名又は名称】 坂口 博

【代理人】

【識別番号】 100091568

【弁理士】

【氏名又は名称】 市位 嘉宏

【代理人】

【識別番号】 100108501

【弁理士】

【氏名又は名称】 上野 剛史

【復代理人】

【識別番号】 100104880

【弁理士】

【氏名又は名称】 古部 次郎

【手数料の表示】

【予納台帳番号】 081504

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9706050

【包括委任状番号】 9704733

【包括委任状番号】 0207860

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 プログラム変換方法、これを用いたデータ処理装置及びプログラム

【特許請求の範囲】

【請求項 1】 コンピュータを制御して、プログラムをコンパイルして機械語コードを生成するプログラム変換方法において、

前記プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態と呼ばれたかについての情報をメモリに格納する第 1 のステップと、

前記プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードと呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第 2 のステップと、

得られた前記基準ではない精度モードと呼ばれた頻度の情報に基づいて、前記所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第 3 のステップと

を含むことを特徴とするプログラム変換方法。

【請求項 2】 前記第 1 のステップでは、前記プログラムのランタイムルーチンにより、前記メソッドが前記基準ではない精度モードと呼ばれた回数を計数してメモリに格納し、

前記第 2 のステップでは、前記メモリに格納されている計数値と予め設定された閾値とを比較し、

前記第 3 のステップでは、前記計数値が前記閾値を超えた場合に前記専用の機械語コードを生成することを特徴とする請求項 1 に記載のプログラム変換方法。

【請求項 3】 前記第 3 のステップでは、コンパイルしようとする前記所定のメソッドに関して、当該メソッドを前記基準ではない精度モードで動作させた方が処理コストを削減させ得る場合に、当該メソッドに対する前記専用の機械語コードを生成することを特徴とする請求項 1 に記載のプログラム変換方法。

【請求項 4】 前記第 3 のステップでは、コンパイルしようとする前記所定

のメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、前記専用の機械語コードを生成することを特徴とする請求項 1 に記載のプログラム変換方法。

【請求項 5】 前記第 3 のステップでは、コンパイルしようとする前記所定のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記専用の機械語コードを生成することを特徴とする請求項 1 に記載のプログラム変換方法。

【請求項 6】 コンピュータを制御して、プログラムをコンパイルして機械語コードを生成するプログラム変換方法において、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第 1 のステップと、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第 2 のステップと、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されているコードに加える第 3 のステップと

を含むことを特徴とするプログラム変換方法。

【請求項 7】 前記第 2 のステップでは、前記呼び出し側メソッドが倍精度モードまたは単精度モードのどちらであるかを調べ、

前記第 3 のステップでは、前記呼び出し側メソッドにおける精度モードと前記対象メソッドにおける精度モードとが異なる場合に前記補助的なコードを生成し追加することを特徴とする請求項 6 に記載のプログラム変換方法。

【請求項 8】 前記第 1 のステップでは、コンパイルしようとする前記対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、機械語コードを生成することを特徴とする請求項 6 に記載のプログラム変換方法。

【請求項 9】 前記第 1 のステップでは、コンパイルしようとする前記対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記領域ごとに演算精度を設定して機械語

コードを生成することを特徴とする請求項 6 に記載のプログラム変換方法。

【請求項 1 0】 プログラムをメソッド単位でコンパイルするコンパイラと

プログラムをインタプリタで実行すると共に前記コンパイラにてコンパイルされたコードで実行するプログラム実行部とを備え、

前記コンパイラは、浮動小数点演算における倍精度モードと単精度モードのうちの基準となる一方の精度モードで前記メソッドの第 1 のコードを生成すると共に、当該メソッドが一定の条件を満足する場合に、倍精度モードと単精度モードのうちの基準ではない精度モードで前記メソッドの第 2 のコードを生成することを特徴とするデータ処理装置。

【請求項 1 1】 前記プログラム実行部は、前記基準ではない精度モードで動作する前記メソッドから前記基準ではない精度モードで動作する他のメソッドを呼び出す場合に、前記コンパイラにて生成された前記第 2 のコードを呼び出すことを特徴とする請求項 1 0 に記載のデータ処理装置。

【請求項 1 2】 前記コンパイラは、コンパイルしようとするメソッドが、前記第 2 のコードの方が処理コストを削減することができ、かつ当該メソッドが前記基準ではない精度モードのメソッドから呼ばれる頻度が高い場合に、前記第 2 のコードを生成することを特徴とする請求項 1 0 に記載のデータ処理装置。

【請求項 1 3】 前記コンパイラは、

コンパイルしようとするメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える過剰精度最適化部と、

前記過剰精度最適化部による処理結果を反映させて前記第 1 のコードまたは前記第 2 のコードを生成するコード生成部と

を備えることを特徴とする請求項 1 0 に記載のデータ処理装置。

【請求項 1 4】 前記コンパイラは、

コンパイルしようとするメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける精度領域解析部と、

前記精度領域解析部による解析結果に基づき、前記領域ごとに演算精度を設定して機械語コードを生成するコード生成部と

を備えることを特徴とする請求項10に記載のデータ処理装置。

【請求項15】 プログラムをコンパイルして機械語コードを生成するデータ処理装置において、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成するコード生成手段と、

前記対象メソッドの精度モードと前記呼び出し側メソッドの精度モードとの関係に基づいて、当該精度モードを合わせるための補助的なコードを生成し、前記コード生成手段にて生成された機械語コードに追加する補助コード追加手段と

を備えることを特徴とするデータ処理装置。

【請求項16】 前記補助コード追加手段は、前記対象メソッドにおける精度モードと前記呼び出し側メソッドにおける精度モードとが異なる場合に、呼び出し側メソッドに対して精度モードを合わせるためのコードを追加することを特徴とする請求項15に記載のデータ処理装置。

【請求項17】 コンパイルしようとする対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えるコード編集手段をさらに備え、

前記コード生成手段は、前記コード編集手段による処理結果を反映させて機械語コードを生成することを特徴とする請求項15に記載のデータ処理装置。

【請求項18】 コンパイルしようとする対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける精度領域解析手段をさらに備え、

前記コード生成手段は、前記精度領域解析手段による解析結果に基づき、前記領域ごとに演算精度を設定して機械語コードを生成することを特徴とする請求項15に記載のデータ処理装置。

【請求項19】 コンピュータを制御して、実行プログラムをコンパイルして機械語コードを生成するプログラムであって、

前記実行プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態と呼ばれたかに

ついでに、前記第 1 の処理と、

前記実行プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードで呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第 2 の処理と、

得られた前記基準ではない精度モードで呼ばれた頻度の情報に基づいて、前記所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第 3 の処理と

を前記コンピュータに実行させることを特徴とするプログラム。

【請求項 2 0】 前記プログラムによる前記第 3 の処理では、コンパイルしようとする前記所定のメソッドに関して、当該メソッドを前記基準ではない精度モードで動作させた方が処理コストが低下する場合に、当該メソッドに対する前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項 1 9 に記載のプログラム。

【請求項 2 1】 前記第 3 の処理では、コンパイルしようとする前記所定のメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項 1 9 に記載のプログラム。

【請求項 2 2】 前記第 3 の処理では、コンパイルしようとする前記所定のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項 1 9 に記載のプログラム。

【請求項 2 3】 コンピュータを制御して、実行プログラムをコンパイルして機械語コードを生成するプログラムであって、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第 1 の処理と、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第 2 の処理と、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されてい

るコードに加える第 3 の処理と

を前記コンピュータに実行させることを特徴とするプログラム。

【請求項 2 4】 前記プログラムによる前記第 3 の処理は、前記対象メソッドにおける精度モードと前記呼び出し側メソッドにおける精度モードとが異なる場合に、呼び出し側のメソッドに対して精度モードを合わせるためのコードを前記コンピュータに追加させることを特徴とする請求項 2 3 に記載のプログラム。

【請求項 2 5】 前記第 1 の処理に先だって、コンパイルしようとする前記対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を、前記コンピュータにさらに実行させることを特徴とする請求項 2 3 に記載のプログラム。

【請求項 2 6】 前記第 1 の処理に先だって、コンパイルしようとする前記対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を、前記コンピュータにさらに実行させると共に、

前記第 1 のステップでは、前記コンピュータに、前記領域ごとに演算精度を設定して機械語コードを生成させることを特徴とする請求項 2 3 に記載のプログラム。

【請求項 2 7】 コンピュータを制御して実行プログラムをコンパイルして機械語コードを生成するプログラムを、当該コンピュータが読み取り可能に記録した記録媒体であって、

前記プログラムは、

前記実行プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態で呼ばれたかについての情報をメモリに格納する第 1 の処理と、

前記実行プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードで呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第 2 の処理と、

得られた前記基準ではない精度モードで呼ばれた頻度の情報に基づいて、前記

所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第3の処理と

を前記コンピュータに実行させることを特徴とする記録媒体。

【請求項28】 コンピュータを制御して実行プログラムをコンパイルして機械語コードを生成するプログラムを、当該コンピュータが読み取り可能に記録した記録媒体であって、

前記プログラムは、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第1の処理と、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第2の処理と、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されているコードに加える第3の処理と

を前記コンピュータに実行させることを特徴とする記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、プログラムのコンパイル技術に関し、特に浮動小数点演算の精度保証を効果的に行う方法に関する。

【0002】

【従来の技術】

J a v a (サン・マイクロシステムズ・インコーポレーテッドの商標)の浮動小数点演算は、単精度演算および倍精度演算それぞれについて精度が言語仕様によって定められている(単精度演算は仮数部24ビット・指数部8ビット、倍精度演算は仮数部53ビット・指数部11ビット)。

一方、I A 3 2 (Intel Architecture-32)アーキテクチャのC P U (いわゆる米国インテル社のx 8 6ファミリ及び互換のC P U)は、この単精度演算及び倍精度演算を行うために、

(方法1) 浮動小数点コントロールワード(以下F C W)の浮動小数点演算モー

ド設定でCPUのモードを切り替えるか、

(方法2) 倍精度モードで演算を行い、単精度の演算結果を得る場合には、浮動小数点レジスタの値をメモリへ単精度でストアし読み出すことにより演算精度を落とす、

といった手法をとる必要があった。

【0003】

ここで、例として単精度演算を行うメソッドを考える。一般にこのメソッドは、倍精度演算のCPUモード(以下、倍精度モード)で呼ばれる場合と、単精度演算のCPUモード(以下、単精度モード)で呼ばれる場合のどちらもあり得る。単精度モードで呼ばれる場合は単精度モードを仮定したコードを実行すれば効率がよい。倍精度で呼ばれる場合は、上述した方法1または方法2により演算の精度保証を行う必要がある。効率の良い方法としては、プログラムを解析して方法1を用いた場合と方法2を用いた場合の処理コストを比較し、より高速な方(オーバーヘッドの少ない方)を実行することができる。

【0004】

JavaにおけるJITコンパイラなどのダイナミックコンパイラで、実行頻度の高いメソッド等を部分的にコンパイルするような分割コンパイル環境の場合、メソッドをコンパイルする際に、当該メソッドが単精度モードと倍精度モードのどちらで呼ばれるかはわからないので、例えばメソッド間の受け渡しは倍精度モードに固定し(すなわち全てのメソッドは倍精度モードで呼ばれる)、単精度演算を行う場合には上記のコスト計算に応じてコード生成を行う。

この種の従来技術としては、例えば下記文献に開示された技術がある。

文献: The Java HotSpot Server Compiler, by M.Paleczny, C.Vick, and C.Click. In Proceeding of USENIX Java Virtual Machine Research and Technology Symposium (JVM '01)

【0005】

【発明が解決しようとする課題】

しかしながら、上記従来の技術は、方法1、2のいずれも処理コストが大きく、多大なオーバーヘッドがかかるという欠点があった。

具体的には、方法1の場合、FCWの切り替え命令(fldcw word ptr [mem])がコストの高い命令である。また、FCWをいったん切り替えると、その範囲内でのメソッド呼び出しの前後では再びFCWの切り替えを行い、基準である倍精度モードに戻す必要があるため、さらなるオーバーヘッドの原因となる。

方法2の場合、単精度演算のたびに、メモリへの書き込み読み出し命令を実行することによってオーバーヘッドを生じ、実行性能の低下を招く。

【0006】

また、そもそもダイナミックコンパイラにおいて、メソッド間解析は、コンパイル時間その他の理由により制限されるため、有効な方法とはいえない。またダイナミックに新しいクラスがロードされるため、いったん解析した結果が無効になるといった無駄が生じる。

そこで、本発明は、浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、オーバーヘッドや解析処理の無駄の発生を回避することを目的とする。

【0007】

【課題を解決するための手段】

上記の目的を達成するため、本発明は、コンピュータを制御して、プログラムをコンパイルして機械語コードを生成する、次のようなプログラム変換方法として実現することができる。すなわち、プログラムを構成するメソッドを実行した際に、このメソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態と呼ばれたかについての情報をメモリに格納する第1のステップと、このプログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、このメソッドが基準ではない精度モードと呼ばれた頻度をメモリに格納されている情報に基づいて調べる第2のステップと、得られた基準ではない精度モードと呼ばれた頻度の情報に基づいて、かかる所定のメソッドに関して、基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第3の

ステップとを含むことを特徴とする。

【0008】

なお、J a v aのようにメソッド間の受け渡しを倍精度モードで行うことが基本となっているシステムでは、基準ではない精度モードは単精度モードとなる。

基準ではない精度モードでの呼び出しの頻度は、例えば、プログラムの実行時に基準ではない精度モードで呼び出された回数を計数しておき、計数値と所定の閾値とを比較することにより判断することができる。

また、専用の機械語コードを生成するのは、コンパイル対象であるメソッドを基準ではない精度モードで動作させた方が処理コストを削減させ得る場合であることが好ましいことは言うまでもない。

さらに、専用の機械語コードを生成する場合に、前処理として、コンパイル対象であるメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を行うことにより、専用の機械語コードを生成する対象メソッドの数が多くなる。あるいは前処理として、コンパイル対象のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を行うことにより、より精度モードの適切なコード生成を行うことができる。

【0009】

本発明による他のプログラム変換方法は、コンパイルしようとする対象メソッドに関し、この対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第1のステップと、この対象メソッドと呼び出し側メソッドとの関係を調べる第2のステップと、得られた関係に応じて、補助的なコードを生成し、メモリに格納されているコードに加える第3のステップとを含むことを特徴とする。

【0010】

ここで、詳細には、第2のステップでは、呼び出し側メソッドが倍精度モードまたは単精度モードのどちらであるかを調べ、第3のステップでは、呼び出し側メソッドにおける精度モードと対象メソッドにおける精度モードとが異なる場合

に精度モードを合わせるための補助的なコードを生成し追加する。

さらに、専用の機械語コードを生成する場合に、前処理として、コンパイル対象であるメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を行うことにより、専用の機械語コードを生成する対象メソッドの数が多くなる。あるいは前処理として、コンパイル対象のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を行うことにより、メソッド単位よりもさらに細かいレベルでプログラムの部分ごとに最適なコード生成を行うことができる。

【 0 0 1 1 】

また、上記の目的を達成する他の本発明は、次のように構成されたデータ処理装置としても実現される。すなわち、このデータ処理装置は、プログラムをメソッド単位でコンパイルするコンパイラと、プログラムをインタプリタで実行すると共にコンパイラにてコンパイルされたコードで実行するプログラム実行部とを備える。このコンパイラは、浮動小数点演算における倍精度モードと単精度モードのうちの基準となる一方の精度モードでメソッドの第1のコードを生成すると共に、このメソッドが一定の条件を満足する場合に、倍精度モードと単精度モードのうちの基準ではない精度モードでメソッドの第2のコードを生成することを特徴とする。

【 0 0 1 2 】

さらに、本発明は、コンパイルしようとする対象メソッドに関し、この対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成するコード生成手段と、この対象メソッドの精度モードと呼び出し側メソッドの精度モードとの関係に基づいて、精度モードを合わせるための補助的なコードを生成し、コード生成手段にて生成された機械語コードに追加する補助コード追加手段とを備えるデータ処理装置としても実現される。

【 0 0 1 3 】

また、本発明は、コンピュータを制御して上述したプログラム変換方法における各ステップに相当する処理を実行させるプログラムや、上述したデータ処理装

置としてコンピュータを機能させるプログラムとしても実現される。このプログラムは、磁気ディスクや光ディスク、半導体メモリ、その他の記録媒体に格納して配布したり、ネットワークを介して配信したりすることにより提供することができる。

【 0 0 1 4 】

【発明の実施の形態】

以下、添付図面に示す実施の形態に基づいて、この発明を詳細に説明する。

本発明では、コンパイル時のコード生成において、浮動小数点演算における精度保証を行う上で、FCW（浮動小数点コントロールワード）のモード設定の切り替えやメモリへの読み書きといったオーバーヘッドを削減するために、次の2つの手法を提案する。すなわち、原則的には単精度モードと倍精度モードのうちの一方を基準としてコード生成を行い、プログラムを実際に動作させて、基準ではないモードで呼ばれることの多いメソッドについて特殊化（specialization）により、他方のモードに対応したコード生成を行う手法（第1の実施の形態）と、呼び出し側（コンパイル対象のメソッドを呼び出したメソッド）の精度モードに従って、当該メソッド単体で最適化するようにコード生成を行う手法（第2の実施の形態）である。

また、本発明では、上記の2つの手法をさらに効果的に実現するため、これらのコード生成の前処理として、メソッド内の演算を置き換える手法（第3の実施の形態）、及びメソッド内のコードを解析する手法（第4の実施の形態）を提案する。

なお、以下の実施の形態では、JavaのJITコンパイラに適用する場合を例として説明する。すなわち、メソッド間の受け渡しは倍精度モードを基準とし、単精度モードでのメソッド呼び出しを取り得る場合に、プログラムの特殊化を行うこととする。

【 0 0 1 5 】

〔第1の実施の形態〕

第1の実施の形態では、実際にプログラムを走らせた上で、所定のメソッドをコンパイルする場合に、当該メソッドについて単精度モードを基準とし、かつ単

精度モードを基準とする他のメソッドからの呼び出しが十分多い場合に、当該メソッドについて特殊化を行う。これにより、FCWの切り替えや精度保証のためのメモリ書き込み読み出し命令のない、効率のよいコードの生成を実現する。

【 0 0 1 6 】

図1は、第1の実施の形態による浮動小数点演算の精度保証方法が実現されるコンピュータシステムの構成を説明する図である。

図1を参照すると、本実施の形態におけるコンピュータシステムは、ソースプログラム（入力コード）をコンパイルするコンパイラ100と、コンパイラ100にてコンパイルされたオブジェクトプログラム（出力コード）を実行して種々の処理を行うプログラム実行部200と、メモリ300とを備える。コンパイラ100及びプログラム実行部200は、パーソナルコンピュータやワークステーションなどのコンピュータシステムにおけるプログラム制御されたCPUにて実現される。メモリ300は、コンピュータ装置のメインメモリであり、RAM等で実現される。メモリ300には、CPUを制御してコンパイラ100として動作させるためのプログラムやコンパイルの対象であるプログラム（以下、実行プログラムと称す）が格納されると共に、この実行プログラムを実行する際に用いられるランタイムルーチン310が格納される。なお、メモリ300に格納されるプログラムは、必要に応じて、適宜磁気ディスクその他の外部記録装置に保存されることは言うまでもない。

【 0 0 1 7 】

図1において、コンパイラ100は、バイトコード（byte code）で記述された入力コードを入力して処理し、機械語で記述された出力コードを生成して出力する。この入力コードの入力は、コード生成装置400にて生成された入力コードを直接入力したり、コード生成装置400にて生成された入力コードを記憶した記憶装置500から入力したり、ネットワーク600上に存在するコード生成装置400や記憶装置500からネットワーク600を介して入力したりすることにより行われる。コンパイラ100により生成された出力コードは、プログラム実行部200により実行される。プログラム実行部200は、コンパイラ100にてコンパイルされた実行プログラムのオブジェクトコード（機械語コード）

を実行するほか、当該実行プログラムをインタプリタによっても実行する。そして、実行プログラムを実行した際に、その実行プロファイルを取得してメモリ 300 に蓄積する。

【0018】

以下、図 2 乃至図 5 を参照して、本実施の形態による操作を詳細に説明する。この操作は、メモリ 300 に格納されたランタイムルーチン 310 にて実現される。

図 2 乃至図 4 は、本実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、図 2 は初期状態を示し、図 3 は最初のコンパイル時の典型的なデータ構造を示し、図 4 は特殊化が実行された場合のデータ構造を示す。また、図 5 は、これらと比較するための従来のランタイムルーチンにより生成されるデータ構造を示す図である。

【0019】

図 2、5 に示すように、プログラムを構成する所定のメソッド 11、12 を実行してメソッド呼び出し（図ではメソッドコールと表記）「call target」を実行する場合、プログラム実行部 200 は、まずメソッドブロック 20 の対応するエントリを参照する。そして、当該エントリに登録された、呼び出そうとする対象のメソッド（以下、対象メソッドと称す）のコードアドレスの登録ポイントから、このエントリに対応するゲートコード 31 と必要な他のランタイムルーチン 40 とを経て、呼び出した対象メソッドをインタプリタまたはコンパイルされたコードで実行する。対象メソッドを実行した後は、呼び出し元であるメソッド 11、12 に戻る。

【0020】

本実施の形態におけるデータ構造を示す図 2 と従来のデータ構造を示す図 5 とを比較すると、本実施の形態では、メソッドブロック 20 における対象メソッドのコードアドレスの登録ポイントとして、上述した通常のエントリ 21（Code Ptr1）の他に単精度専用のエントリ 22（Code Ptr2）が設けられている。そして、呼び出し側メソッド 11、12 のコールサイトにおける現在の FCW のモードによって、通常のエントリ 21 または単精度専用のエントリ 22 を呼び分ける。

すなわち、単精度モードを基準とするメソッド 1 2 から対象メソッドを呼び出す場合、この新たに設けられた単精度専用のエントリ 2 2 を参照することとなる。

また、このエントリ 2 2 に対応するゲートコード 3 2 として、

- 1) F C W を倍精度に設定し (fldcw f2d)、
- 2) 専用カウンタ (単精度モードカウンタ) をインクリメントし (inc counter)、
- 3) 通常のゲートコードエントリへジャンプする (jmp Gate)

コードを用意する。

【 0 0 2 1 】

また、単精度モードでのメソッド呼び出しでは、単精度専用コードへジャンプすることを保証しない。すなわち、呼び出した対象メソッドが倍精度モードを基準とするメソッドである可能性がある。そこで、メソッド 1 2 に示すように、呼び出し命令 (call target) の直後、すなわち呼び出した対象メソッドの実行後に戻るアドレスに、F C W を単精度モードへ再設定する命令 (fldcw d2f) を生成する。対象メソッドが倍精度モードと単精度モードのどちらの状態と呼ばれたかについての情報を取得する精度モード情報取得手段としての単精度モードカウンタの値は、メモリ 3 0 0 に保持され、後述するコンパイル時にコンパイラ 1 0 0 により参照される。

【 0 0 2 2 】

図 5 を参照すると、単精度モードでのメソッド呼び出しでは、メソッド間の受け渡しを倍精度モードに固定する原則にしたがって、メソッド 1 2 に示すように、呼び出し命令 (call target) の前で F C W を倍精度モードに切り替え (fldcw f2d) 呼び出し命令 (call target) の後ろで単精度モードに戻していることがわかる (fldcw d2f)。

図 2 に示した状態では、単精度モードの下でのメソッド呼び出しは、ゲートコード 3 2 によって、従来と同様にメソッドの境界で倍精度モードへと設定され、通常のパスを経て呼び出しポイントに戻ってきた後、再度、単精度モードへ再設定される。しかし、この状態のままでは、単精度モードからの呼び出しのパスには、ゲートコード 3 2 におけるカウンタのインクリメントやジャンプ命令などの

オーバーヘッドがかかることになる。

【 0 0 2 3 】

次に、メソッド 1 1、1 2 から呼び出されるメソッドがコンパイルされる場合を考える。プログラムの実行が繰り返され、所定のメソッドの実行回数を計数するカウンタ値が予め定められた閾値を超えた場合に、コード生成手段であるコンパイラ 1 0 0 により当該対象メソッドがコンパイルされてオブジェクトコードが生成され、当該対象メソッドの実行がインタプリタにおける実行からコンパイラにおける実行へ遷移する。

コンパイラ 1 0 0 によるコンパイル時に、メモリ 3 0 0 に保持されている単精度モードカウンタの値をチェックし、インタプリタからコンパイラへ遷移するためのカウンタ値と比較することによって、呼び出し側における F C W のモードは単精度と倍精度のどちらの設定が多いかを知ることができる。

単精度モードカウンタの値が小さい場合、メソッド境界は倍精度モードであると仮定した従来と同様のオブジェクトコード (general version) 5 0 を生成する。図 3 は、この時のデータ構造を示している。

このオブジェクトコード 5 0 は、メソッドブロック 2 0 における通常のメソッドのコードアドレスのエントリ 2 1 に登録される。また、単精度モード専用のエントリ 2 2 に対応したゲートコード 3 2 内のジャンプアドレスのターゲットとなるように、このゲートコード 3 2 におけるジャンプ命令が書き換えられる。

【 0 0 2 4 】

さらに、対象メソッドが実行時に頻繁に呼び出される (ホットな) メソッドである場合の再コンパイルを考える。

この対象メソッドの呼び出しに関して、単精度専用のエントリ 2 2 に対応するゲートコード 3 2 における単精度モードカウンタの値が十分大きい場合は、F C W が単精度モード設定の状態で (すなわちメソッド 1 2 から) 呼び出されたケースの頻度が高いと考えられる。このメソッドが単精度を基準モードとしてコンパイルする条件を満たす時、単精度モードでの直接呼び出しに特殊化されたバージョンのための再コンパイルを行うこととなる。かかる条件は任意に定め得るが、例えば、単精度モードでの呼び出し回数が倍精度モードでの呼び出し回数よりも

多い場合などというように、適当な閾値を設定することができる。図4は、再コンパイルにより、単精度モードでの直接呼び出しに用いられる特殊化されたオブジェクトコード (special version) 60を生成した場合のデータ構造を示している。

【0025】

生成された特殊化されたオブジェクトコード60は、メソッドブロック20における単精度専用のコードアドレスのエントリ22へ登録する。また、ゲートコード32に対しては、この新しいオブジェクトコード60へ直接ジャンプするように先頭コードを無条件ジャンプ命令に書き換える。これによって、単精度モードの仮想的なコールサイト (virtual call site) では、自動的に新しく登録されたコードを呼び出すことが可能となる。また、静的メソッド (static) や非仮想的なコールサイト (nonvirtual call site) で直接ゲートコード32を呼んでいたところは、既存のバックパッチのメカニズムによって、次の呼び出しから直接オブジェクトコード60を呼び出すことになる。

最後に、この単精度モード専用のオブジェクトコード60は、リターンアドレスのポインタを固定長バイト分だけスライドさせる。これによって、呼び出し側のメソッド12における呼び出し命令 (call target) の直後に置かれている、FCWを単精度モードに再設定するための命令 (fldcw d2f) をスキップする。

【0026】

上述したように、まず通常のコンパイルを行った後、メソッドの実行状態に応じて再コンパイルする際に特殊化されたオブジェクトコード60を生成する手順の他に、図2に示した初期状態から初めて対象メソッドがコンパイルされる時に、単精度モードカウンタの値がすでに十分大きい時 (例えばインタプリタにおける実行からコンパイラにおける実行へ遷移するための閾値の90%を超える場合など) には、倍精度モードでの呼び出しを仮定した通常のオブジェクトコード50を生成することなく、初めから単精度用の特殊化されたオブジェクトコード60を生成するようにしても良い。この場合、通常のコンパイル (倍精度モードでの呼び出しを仮定したオブジェクトコード50の生成) を後で駆動させる可能性を残すため、インタプリタにおける実行からコンパイラにおける実行への遷移の

ためのカウンタに対し、その値を調整する（例えば閾値からFモードカウンタを引いた値に再設定する）ことが必要になる。

【0027】

図6は、以上の図2乃至図4のデータ構造を参照して説明した、本実施の形態によるプログラムの特殊化の行程を示すフローチャートである。

図6に示すように、所定のメソッド（上記の説明における対象メソッド）の呼び出し頻度が十分に大きく（ホットメソッドを検出）、当該メソッドをコンパイルする場合、コンパイラ100は、まずゲートコード32の単精度モードカウンタの値が閾値を超えたかどうか（予め設定された条件を満たしたかどうか）を調べる（ステップ601、602）。

単精度モードカウンタの値が閾値を超えた場合は、次に当該メソッドが単精度モードを基準としてコンパイル可能かどうかを判断する（ステップ603）。これは、当該メソッド中の倍精度演算、単精度演算、メソッド呼び出し命令の数を数えて、当該メソッド全体について、単精度モードを基準とし必要に応じてFCWを倍精度モードへ切り替えることと、倍精度モードを基準とし必要に応じてFCWを単精度モードへ切り替える（またはメモリを用いた精度保証を行う）こととのどちらが低コストかを解析することによって判断できる。

【0028】

処理対象であるメソッドが単精度モードを基準としてコンパイル可能と判断された場合、次にコンパイラ100は、単精度モードでの呼び出しに特殊化されたオブジェクトコード60を生成する（ステップ604）。そして、メソッドブロック20の単精度専用コードアドレスのエントリ22に登録し（ステップ605）、単精度専用のゲートコード32において、fldcw f2d及びinc counterをnopに変換し（削除し）、ジャンプアドレスをオブジェクトコード60のアドレスに書き換えて処理を終了する（ステップ606）。

【0029】

一方、ステップ602で単精度モードカウンタの値が閾値を超えていない場合、またはステップ603で当該メソッドが単精度モードを基準としてコンパイル可能ではないと判断された場合、コンパイラ100は、倍精度モードでの呼び出

しを仮定した（すなわち通常の）オブジェクトコード 5 0 を生成し（ステップ 6 0 7）、メソッドブロック 2 0 における当該メソッドのコードアドレスのエントリ 2 1 に登録する（ステップ 6 0 8）。

次に、当該メソッドに関する単精度専用のオブジェクトコード 6 0 が既に生成されているかどうかを調べ（ステップ 6 0 9）、かかるオブジェクトコード 6 0 が生成されていないならば、以後は新たに生成されたオブジェクトコード 5 0 を使用するため、単精度専用のゲートコード 3 2 におけるジャンプアドレスをゲートコード 3 1 からオブジェクトコード 5 0 のアドレスに書き換えて処理を終了する（ステップ 6 1 0）。一方、単精度専用のオブジェクトコード 6 0 が既に生成されているならば、単精度モードでの呼び出しに対しては当該オブジェクトコード 6 0 を使用できるため、そのまま処理を終了する。

【 0 0 3 0 】

以上のように、第 1 の実施の形態によれば、倍精度モードを基準としつつも、単精度モードのメソッドからの呼び出し頻度の高いメソッドに対して、単精度専用の特殊化されたコードを生成する。そして、当該単精度モードのメソッドから当該特殊化されたコードを直接呼び出すことにより、FCW による精度モードの切り替えやメモリへの書き込み読み出しといった、処理コストの高い冗長な処理を削減することができる。

また、プログラムの実行履歴に基づき、単精度モードのメソッドからの呼び出し頻度の高いメソッドに関して、この特殊化されたコードを生成するため、不必要なコードサイズの増加を招来することはない。

【 0 0 3 1 】

〔第 2 の実施の形態〕

第 2 の実施の形態では、コンパイル対象のメソッドは呼び出し側の精度モードで呼び出される。メソッドをコンパイルする際、コンパイラは、当該メソッドの開始位置と終了位置とにおける精度を、呼び出し精度と想定する。生成されたコードは想定された精度で実行開始され、終了する際には当該精度で終了するものとする。

第 2 の実施の形態による浮動小数点演算の精度保証方法は、第 1 の実施の形態

と同様に、図 1 に示すように構成されたコンピュータシステムにて実現される。

【 0 0 3 2 】

単精度の演算を行うプログラムでは、単精度演算を行うメソッド同士が呼び合うことが多い。そこで、単精度モードで実行するメソッドが別の単精度演算を行うメソッドを呼ぶ場合（この状況を、以下、対象状況と称す）、FCWによる精度モードの切り替えやメモリの書き込み読み出しを行わずに、そのまま実行できればオーバーヘッドを回避することができる。

J a v a のように、メソッド間の受け渡しにおいて倍精度モードを基準とする場合、対象状況においては、メソッド A がメソッド B を呼ぶ際の、

- a) メソッド A における単精度から倍精度へのモード変更、
 - b) メソッド B における倍精度から単精度へのモード変更、
- 及びメソッド B がメソッド A に戻る際の、
- c) メソッド B における単精度から倍精度へのモード変更、
 - d) メソッド A における倍精度から単精度へのモード変更、
- の 4 つの冗長なモード変換によるオーバーヘッドが、回避したいオーバーヘッドである。

本実施の形態では、上述したように、呼び出し側の精度モードでメソッドを呼ぶ。したがって、同一メソッドが複数の精度から呼び出される場合は、同一メソッドに対して精度に対応して複数のコードが生成されてしまう。コードサイズやコンパイル時間などでそれを抑制する必要がある場合、一度、いずれかの精度でメソッドがコンパイルされたら、それ以降は異なる精度で呼ぶ場合であっても、当該異なる精度に対応するコードを生成しないこともあり得る。その場合、精度を合わせるためのコードを経由してターゲットコードを呼ぶこととなる。また、この部分に関して上述した第 1 の実施の形態を適用しても良い。

【 0 0 3 3 】

上述したように本実施の形態において、コード生成手段であるコンパイラ 1 0 0 は、メソッドをコンパイルする際、メソッドが呼ばれる時の精度モード（すなわちゴールサイトの精度モード）で呼び出しが行われると想定し、コンパイル対象である当該メソッド単体を最も最適化するようにコード生成を行う。呼び出し

精度が倍精度ならば倍精度モードを仮定してコンパイルし（このコンパイルで生成されるコードを倍精度コードと称す）、呼び出し精度が単精度ならば単精度モードを仮定してコンパイルする（このコンパイルで生成されるコードを単精度コードと称す）。そして、倍精度演算と単精度演算とが混在するならば、精度モードの切り替え（方法1）やメモリへの書き込み読み出し（方法2）を用いた精度保証の手法を組み合わせるコンパイルする。

【0034】

またコンパイラ100は、コンパイルしたメソッドに関し、当該コードの入りで仮定している精度モードに応じて、単精度コードか倍精度コードかの種別（デフォルト精度モード）を当該メソッドのコード中やランタイムルーチンに登録しておく。さらに、コンパイルしたメソッドに対して精度モードの切り替え（方法1）が行われた場合、精度モードが変化している区間とその精度モードを同様に登録する。

【0035】

コンパイルしたコードが単精度コードしか存在せず、インタプリタで実行されるメソッドから呼ばれる可能性がある場合、インタプリタは常に倍精度モードで動いていると仮定し、呼び出しの前後で精度モードを切り替える。そして、当該コンパイルされたコードを実行する前に一旦単精度モードに切り替え、当該コードから戻ったときに再度倍精度モードに戻す。

【0036】

所定のメソッドに着目するとき、このメソッドは静的メソッド（static method）である場合と仮想メソッド（virtual method）である場合とに分けられる。また、それぞれの場合で、メソッドに対して単精度コードが生成される場合と、倍精度コードが生成される場合とがある。当該メソッドが仮想メソッドでありかつ仮想的に呼び出される場合は、単精度コードから呼び出される場合と倍精度コードから呼び出される場合の両方があるため、当該メソッドに関して、単精度コードから呼び出された場合のコードと倍精度コードから呼び出された場合のコードとがそれぞれ生成され、プログラムの実行状況に応じて選択されることとなる。

【 0 0 3 7 】

次に、コンパイラ 1 0 0 のコード生成について、具体的な場合に分けて、さらに説明する。

コンパイラ 1 0 0 による分割コンパイル環境では、コンパイル対象であるメソッドと他のメソッドとの間で、次の 2 つの状況が生じ得る。

- 1 : メソッド A が既にコンパイルされて、今メソッド B をコンパイルする。
- 2 : メソッド A がまだコンパイルされず、今メソッド B をコンパイルする。

【 0 0 3 8 】

1 の状況に対して、コンパイラ 1 0 0 は次のようにコード生成を行う。

メソッド A のコールサイトが倍精度コードである場合、メソッド B の演算が倍精度演算のみならば倍精度でコンパイルし（倍精度コード）、単精度演算のみならば単精度でコンパイルする（単精度コード）。倍精度演算と単精度演算とが混在しているならば、処理コストを計算した上で倍精度コードまたは単精度コードでコンパイルし、必要に応じて精度モードの切り替えやメモリへの書き込み読み出しといった既存の手法により精度保証を行うか、または後に説明する第 4 の実施の形態を適用する。

一方、メソッド A のコールサイトが単精度コードである場合、メソッド B の演算が単精度演算のみならば単精度でコンパイルする（単精度コード）。それ以外ならば、倍精度コードでコンパイルし、必要に応じて精度モードの切り替えとメモリへの書き込み読み出しとを組み合わせる既存の手法により精度保証を行うか、または後に説明する第 4 の実施の形態を適用する。

図 7 は、以上の場合におけるコード生成を概念的に説明する図である。

【 0 0 3 9 】

2 の状況に対して、コンパイラ 1 0 0 は次のようにコード生成を行う。

メソッド B の演算が倍精度演算のみならば倍精度でコンパイルし（倍精度コード）、単精度演算のみならば単精度でコンパイルする（単精度コード）。倍精度演算と単精度演算とが混在しているならば、処理コストを計算した上で倍精度コードまたは単精度コードでコンパイルし、必要に応じて精度モードの切り替えとメモリへの書き込み読み出しとを組み合わせる既存の手法により精度保証を行う

。倍精度コードと単精度コードのどちらを生成するかは、精度モードの切り替えとメモリへの書き込み読み出しとの組み合わせ方に依存する。または後に説明する第4の実施の形態を適用しても良い。

メソッドAのコールサイトはインタプリタで実行されており、通常インタプリタは精度モードを決めて実行されるので、コンパイルコードを呼ぶためのglueコードで、必要に応じて精度モードを切り替える。すなわち、インタプリタが倍精度モードで実行されている場合、メソッドBが倍精度コードであれば精度モードの切り替えを行わずにコンパイルされたコードを呼び出す。一方、メソッドBが単精度コードであれば、glueコードにて倍精度モードから単精度モードに切り替えてコンパイルされたコードを呼び出し、メソッドAに戻る際に倍精度モードに戻る。

図8は、以上の場合におけるコード生成を模式的に説明する図である。

【0040】

図9は、上述したコンパイルの手順を説明するフローチャートである。

図9に示すように、コンパイラ100は、ホットメソッドを検出して所定のメソッドをコンパイルする場合、まず、コンパイルしようとするメソッド（以下、対象メソッドと称す）内の演算の種類（単精度演算のみか、倍精度演算のみか、両者が混在するか）に応じてコードを生成し、メモリ300に一時的に格納する（ステップ901、902）。次に、コンパイラ100は、対象メソッドと当該対象メソッドを呼び出す他のメソッド（呼び出し側メソッド）との関係を検討し（ステップ903）、得られた関係に応じて、図7及び図8を参照して説明したように、補助的なコードを生成し、メモリ300に格納されているステップ902で生成されたコードに加える。すなわち、必要な精度モード切り替えのためのコードを対象メソッドのコードや上記他のメソッドのコードに挿入したり、呼び出し側メソッドがコンパイルされていない場合はglueコードに挿入したりする（ステップ904）。

【0041】

以上のように、第2の実施の形態によれば、所定のメソッドをコンパイルする場合に、呼び出し側（当該メソッドを呼び出したメソッド）の精度モードで開始

し終わることを前提とし、コンパイル対象であるメソッド単体で最適化されるようにコード生成を行う。そのため、所定の精度モード（例えば倍精度モード）を基準とすることによって発生する冗長な精度モードの切り替えを削減することができる。

【0042】

〔第3の実施の形態〕

上記第1の実施の形態では、実際の動作において単精度で呼ばれることの多いメソッドに対して単精度を基準としたコードを生成し、第2の実施の形態では、メソッド内の演算の種類に応じて演算精度を設定してコードを生成した。しかしながら、プログラム中の個々のメソッドにおいては、単精度で十分な演算でありながら、倍精度で演算を行うように記述されているために倍精度モードでコードが生成されてしまうメソッドや、大部分が単精度の演算であるが、倍精度が必要な演算が部分的に現れるメソッドが存在する。このようなメソッドについては、第1、第2の実施の形態においても単精度モードでのコード生成を行うことができず、プログラムの実行効率の向上が抑制されてしまう。

そこで、第3の実施の形態は、メソッドをコンパイルする場合に、前処理としてメソッド内の演算を置き換えることにより、第1、第2の実施の形態によるコード生成の効果をさらに向上させる。

【0043】

本実施の形態では、単精度で十分な演算でありながら、倍精度で演算を行うように記述されているために倍精度モードでコードが生成されてしまうメソッドを対象として、当該メソッド内の演算の置き換えを行う。このコードの編集処理を過剰精度最適化（XP0；excessive-precision optimization）と呼ぶ。

過剰な倍精度モードでコードが生成されてしまう原因としては、Javaの提供するメソッドAPIが倍精度の演算用の関数のみを提供したり、プログラマが過剰な精度を指定してメソッドを記述したりする等、様々な原因が考えられる。本実施の形態は、演算結果の精度ビット数を、関数の定義（definition）と使用（use）との関係（def-use関係）に基づいて伝搬する。そして、単精度演算で同じ結果を得られるならば、当該メソッド中の倍精度演算を単精度演算に置き換え

る。これにより、演算精度を落とすための無駄なメモリ書き込みを除去する。

【 0 0 4 4 】

第 3 の実施の形態による浮動小数点演算の精度保証のための過剰精度最適化は、第 1 の実施の形態と同様に、図 1 に示すように構成されたコンピュータシステムにて実現される。

図 1 0 は、本実施の形態におけるコンパイラ 1 0 0 の機能を説明するブロック図である。

図 1 0 に示すように、本実施の形態のコンパイラ 1 0 0 は、コンパイル対象のメソッドに対して過剰精度最適化の処理を行う過剰精度最適化部 1 1 0 と、過剰精度最適化処理を施されたメソッドのバイナリコードを機械語コードに変換するコード生成部 1 2 0 とを備える。過剰精度最適化部 1 1 0 及びコード生成部 1 2 0 は、メモリ 3 0 0 に格納されているプログラムにて制御された CPU にて実現される仮想的なソフトウェアブロックである。なお図 1 0 には、本実施の形態における特徴的な構成のみを記載している。実際には、図示の構成の他に、コンパイル対象のメソッドのバイナリコードを構文解析する手段や、本実施の形態による過剰精度最適化以外の種々の最適化処理を実行する手段が設けられることはいうまでもない。

【 0 0 4 5 】

コード編集手段である過剰精度最適化部 1 1 0 は、コード生成部 1 2 0 によるコード生成の前処理として、コンパイル対象のメソッド中から過剰精度（倍精度）の演算を行う部分を検出し、適切な低い精度（単精度）の演算に置き換える。具体的には、まずメソッド中の倍精度の演算（関数）に対して個別に過剰精度か否かの判定を行い、過剰精度と判断された演算を単精度に置き換える。ここで、コンパイル時にデータフロー解析を行い、検出された全ての倍精度による演算に対して機械的に過剰精度か否かの判定を行っても良いし、メソッドの実行によりメモリ 3 0 0 等に蓄積される実行時プロファイルを参照して、頻繁に実行される倍精度の演算を対象として過剰精度か否かの判定を行っても良い。

【 0 0 4 6 】

ある浮動小数点計算 C は、次の 2 つの条件を満足する場合に、過剰精度計算で

あると判断される。すなわち、

(1) 関数 F (計算 C における入力を処理し出力を結果とするコード列、1 命令コードの演算や別メソッドを呼び出す関数を含む) は、入力値がある制限された値 (例えば仮数部を単精度で表現可能) である場合に、より悪い精度モード (単精度モード) で計算しても同じ結果を得られる。

(2) 入力値は、全て上の制限された値である。

本実施の形態では、このような条件に合致する浮動小数点計算を集めた過剰精度計算表 3 2 0 を用意し、過剰精度最適化部 1 1 0 による過剰精度か否かの判定に用いる。図 1 1 は、過剰精度計算表 3 2 0 の例を示す図である。図示のように、過剰精度計算表 3 2 0 には、上記条件 (1) を満たす浮動小数点計算の種類 (計算種別) ごとに、この条件 (1) に合致するための入力値の制限が登録されている。この過剰精度計算表 3 2 0 は、過剰精度となることが予め経験的に分かっている浮動小数点計算について作成され、メモリ 3 0 0 に格納されているものとする。

【 0 0 4 7 】

図 1 2 は、過剰精度最適化部 1 1 0 が所定の計算 C に関して過剰精度か否かを判定する処理の流れを説明するフローチャートである。過剰精度最適化部 1 1 0 は、コンパイル対象であるメソッド中の各浮動小数点計算に対して、図 1 2 に示す以下の処理を行う。

まず、所定の浮動小数点計算 C に着目する (ステップ 1 2 0 1)。そして、この計算 C における関数 F が過剰精度計算表 3 2 0 に登録されているかどうかを調べる (ステップ 1 2 0 2)。登録されているならば、次に、当該関数 F の入力値の全てが過剰精度計算表 3 2 0 に登録されている制限に該当するかどうかを判断する (ステップ 1 2 0 3)。該当するならば、着目中の計算 C は過剰精度計算であると判定する (ステップ 1 2 0 4)。一方、関数 F が過剰精度計算表 3 2 0 に登録されていないか、または関数 F の入力値のいずれかが過剰精度計算表 3 2 0 に登録されている制限に合致しない場合は、着目中の計算 C は過剰精度計算ではないと判定する (ステップ 1 2 0 2、1 2 0 3、1 2 0 5)。

【 0 0 4 8 】

図 1 3 は、過剰精度計算と判定された計算 C を低精度（単精度）の演算に置き換える処理の流れを説明するフローチャートである。過剰精度最適化部 1 1 0 は、コンパイル対象であるメソッド中の各浮動小数点計算に対して、図 1 3 に示す以下の処理を行う。

まず、着目中の計算 C が過剰精度計算か否かを判断する（ステップ 1 3 0 1）。これは図 1 2 に示した処理による判定結果に基づく。計算 C が過剰精度計算であるならば、過剰精度最適化部 1 1 0 は、演算 F を低精度の演算に置換し、入力から精度変換の命令を削除し、定数を低精度で同じ値を表す定数表現に変換する（ステップ 1 3 0 2）。そして、入力に対する出力値の符号、仮数部、指数部の値がそれぞれどのような値を取り得るかに関する情報（以下、値情報）を生成し、演算 F（特に出力を行うコード）に対して属性情報として登録する（ステップ 1 3 0 3）。

【 0 0 4 9 】

以上のようにして、過剰精度最適化部 1 1 0 による処理が終了する。この過剰精度最適化処理の結果、すなわち過剰精度計算の単精度化がなされたメソッドは、メモリ 3 0 0 に保持され、コード生成部 1 2 0 による機械語コードの生成に用いられることとなる。

【 0 0 5 0 】

図 1 4 及び図 1 5 は、具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図である。図 1 4 は、単精度浮動小数点値に値 0.5 を乗算する倍精度計算を単精度化する例を疑似コードで示し、図 1 5 は、単精度浮動小数点値の平方根 $\text{sqrt}(v)$ を倍精度で求めて単精度に丸める計算を単精度化する例を疑似コードで示す。

図 1 4 において、f は単精度浮動小数点値を結果として残す任意の計算である。L F 1、L F 2、L F 3 は単精度浮動小数点値を持つ変数である。L D 1、L D 2、L D 3 は倍精度浮動小数点値を持つ変数である。F 2 D は単精度浮動小数点値を倍精度浮動小数点値に変換する命令である。F L O A D は単精度浮動小数点の定数をロードする命令、D L O A D は倍精度浮動小数点の定数をロードする命令である。F M U L は単精度浮動小数点の乗算する命令、D M U L は倍精度浮

動小数点の乗算する命令である。

処理対象の倍精度計算において、変数LD2に命令F2Dと共に変数LF1が代入され、変数LD3に値0.5の倍精度浮動小数点値をロードする命令DLOADが代入され、2つ目の変数LD2に倍精度の乗算命令DMULが代入されている。

【0051】

この倍精度の浮動小数点計算に対して、過剰精度最適化部110による判定（図12参照）が行われた場合を考える。このとき図11に示した過剰精度計算表320が参照されるものとする。

すると、命令DMULは、過剰精度計算表320に登録されており、入力である変数LD2の実体が単精度浮動小数点値を持つ変数LF1であるので入力1の制限に該当する。そこで、過剰精度最適化部110はこの倍精度計算を過剰精度計算であると判定する。そして、倍精度計算を単精度計算に置き換える（図13参照）。

図14の例では、精度変換の命令LD2=F2Dが削除され、命令DLOADが値0.5の単精度浮動小数点値をロードする命令FLOADに変換され（LD3→LF3）、さらに命令DMULが単精度の乗算命令FMULに変換される（LD2→LF2）。

以上のようにして、過剰精度最適化部110による単精度化（過剰精度最適化処理）の処理が終了する。

【0052】

図15において、f、LF1、LF2、LF3、LD1、LD2、F2Dは、図14の場合と同様である。D2Fは、倍精度浮動小数点値を単精度浮動小数点値に変換する命令である。FSQRTは単精度浮動小数点値の平方根を計算する命令、DSQRTは倍精度浮動小数点の平方根する命令である。

処理対象の倍精度計算において、1つ目の変数LD2に命令F2Dと共に変数LF1が代入され、2つ目の変数LD2に1つ目の変数LD2の倍精度浮動小数点値の平方根を求める命令DSQRTが代入され、変数LF3に2つ目の変数LD2の結果を単精度浮動小数点値に変換する命令D2Fが代入されている。

【 0 0 5 3 】

この倍精度の浮動小数点計算に対して、過剰精度最適化部 1 1 0 による判定（図 1 2 参照）が行われた場合を考える。このとき図 1 1 に示した過剰精度計算表 3 2 0 が参照されるものとする。

すると、命令 D S Q R T と命令 D 2 F との組合せは、過剰精度計算表 3 2 0 に登録されており、入力である 1 つ目の変数 L D 2 の実体が単精度浮動小数点値を持つ変数 L F 1 であるので入力 1 の制限に該当する。そこで、過剰精度最適化部 1 1 0 はこの倍精度計算を過剰精度計算であると判定する。そして、倍精度計算を単精度計算に置き換える（図 1 3 参照）。

図 1 5 の例では、精度変換の命令 $L D 2 = F 2 D$ が削除され、命令 D S Q R T と命令 D 2 F との組が単精度浮動小数点値の平方根を求める命令 F S Q R T に変換される（ $L D 2 + L F 3 \rightarrow L F 3$ ）。

以上のようにして、過剰精度最適化部 1 1 0 による単精度化（過剰精度最適化処理）の処理が終了する。

【 0 0 5 4 】

以上のようにして、本実施の形態ではメソッド中の過剰精度の計算を単精度の計算に置き換えるため、例えば第 1 の実施の形態と共に本実施の形態を用いれば、過剰精度の計算の単精度化により、単精度用の特殊化されたコードの生成対象となるメソッドが多くなる。また、第 2 の実施の形態と共に本実施の形態を用いれば、過剰精度の計算の単精度化により、当該メソッドにおいて、より精度モードの適切なコード生成を行うことが可能となる。

【 0 0 5 5 】

なお、本実施の形態では、過剰精度計算表 3 2 0 を予め作成してメモリ 3 0 0 に格納しておくこととしたが、過剰精度計算表 3 2 0 のエントリを動的に追加していくことも可能である。すなわち、図 1 2 を参照して説明した判定処理において、ステップ 1 2 0 2 で所定の計算 C における関数 F が過剰精度計算表 3 2 0 に登録されていないと判断された場合に、当該計算 C が上述した過剰精度計算であるか否かを判断する条件（1）（2）に合致するか（入力値の全集合を使って計算 C から得た結果と、計算 C を低精度で実行して得た結果が同じであるか）を実

際に計算して調べ、合致するならば関数 F 及びその入力値を過剰精度計算表 3 2 0 に登録する。言い換えれば、メソッドの実行時に徐々に過剰精度計算表 3 2 0 を構築する方法である。なお、計算 C が条件 (1) (2) に合致すると判断した場合、過剰精度計算表 3 2 0 に登録すると共に図 1 3 に示した置き換え処理に移行しても良いし、過剰精度計算表 3 2 0 を作成する処理と図 1 2 の判定処理とを別個の処理として、当該計算 C を対象として改めて図 1 2 の判定処理を行っても良い。ただし、計算 C が条件 (1) (2) に合致するかどうかを判断する処理は多大な時間 (処理コスト) を要するため、特に重要な計算に限定して用いたり、この処理コストがプログラムの実行におけるオーバーヘッドとならない静的コンパイラに用いたりすることが好ましい。

【 0 0 5 6 】

〔第 4 の実施の形態〕

次に、第 4 の実施の形態は、単精度の演算と倍精度の演算とが混在するメソッドを対象として、倍精度演算を行うコード領域を孤立させることにより、第 1、第 2 の実施の形態によるコード生成の効果をさらに向上させる。

本実施の形態では、処理対象であるメソッドのコード解析を行ってメソッド中のコードを単精度演算だけを含む領域 (単精度領域) とそれ以外の領域とに分け、倍精度による演算が必要なコード領域を孤立させる。この処理を精度領域解析 (PRA; precision region analysis) と呼ぶ。

経験的に、同一メソッドが単精度演算と倍精度演算の両者を含む場合でも、単精度演算と倍精度演算が入り組んで (例えば交互に) 現れることはなく、単精度演算がある程度連続するコード領域が見られる。そこで、本実施の形態は、当該メソッド中のコードを制御フローに従って調べ、単精度演算を含みかつ倍精度演算を含まない連続する単精度領域を探索する。単精度領域が得られたならば、当該単精度領域に対して個別に CPU の精度モードを設定することにより、メソッド単位よりもさらに細かいレベルで精度モードの設定を行う。

【 0 0 5 7 】

第 4 の実施の形態による浮動小数点演算の精度保証のための精度領域解析は、第 1 の実施の形態と同様に、図 1 に示すように構成されたコンピュータシステム

にて実現される。

図16は、本実施の形態におけるコンパイラ100の機能を説明するブロック図である。

図16に示すように、本実施の形態のコンパイラ100は、コンパイル対象のメソッドに対して精度領域解析の処理を行う精度領域解析部130と、精度領域解析処理を施されたメソッドのバイナリコードを機械語コードに変換するコード生成部120とを備える。精度領域解析部130及びコード生成部120は、メモリ300に格納されているプログラムにて制御されたCPUにて実現される仮想的なソフトウェアブロックである。なお図16には、本実施の形態における特徴的な構成のみを記載している。実際には、図示の構成の他に、コンパイル対象のメソッドのバイナリコードを構文解析する手段や、本実施の形態による過剰精度最適化以外の種々の最適化処理を実行する手段が設けられることはいうまでもない。

【0058】

精度領域解析部130は、コード生成部120によるコード生成の前処理として、コンパイル対象のメソッドに対し、制御フローにしたがってコード解析を行い、可能な限り大きな範囲となるように単精度領域を探索する。そして、得られた単精度領域に対して個別にCPUの精度モードの設定を行う。具体的には、各単精度領域に対し、浮動小数点演算における精度保証を行う方法として、CPUの精度モードを切り替える方法（方法1）またはメモリへの書き込み読み出しを用いて演算精度を落とす方法（方法2）のどちらを用いた場合に処理コストが低いかを調べる。そして、方法1を取った方が処理コストが低いならば、当該単精度領域の入り口と出口で精度モード切り替えを行い、当該単精度領域の処理を単精度モードで実行することとする。一方、方法2を取った方が処理コストが低いならば、当該単精度領域の処理を倍精度モードで実行することとする。

【0059】

図17、18は、精度領域解析部130が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

図 1 7、1 8 に示すように、精度領域解析部 1 3 0 は、まず、処理対象として、異なる精度の演算が混在するメソッドのコード（バイナリコード）を入力する。そして、制御フロー上の基本ブロック（制御フローが途中に入ることもなく、途中から出ることもないようなコード列の範囲）を、実行頻度を考慮した深さ優先順により順序づけ、順序 0 とする（ステップ 1 7 0 1）。例えば、図 1 9（A）に示すソースコードを考えると、これに対して図 1 9（B）に示す 4 つの基本ブロックができ、深さ優先で順序づけると、基本ブロック 1、基本ブロック 2、基本ブロック 4、基本ブロック 3 の順となる。

【0 0 6 0】

また、メソッドの先頭を領域（単精度領域またはその他の領域）R の開始位置とし、当該メソッドが呼ばれた際の精度（例えば、第 1 の実施の形態では実際にプログラムを走らせて決定された精度モード、第 2 の実施の形態ではメソッドごとに最適化されるように決定された精度モード）を現在の精度 P（領域 R に設定される精度モード）とする（ステップ 1 7 0 2）。領域 R 及びその精度 P は、例えば図 2 0 に示すような管理テーブル（データ構造）を用意し、これに登録して管理する。この管理テーブル 2 0 0 1 は、コンパイラ 1 0 0 の精度領域解析部 1 3 0 にて生成され、メモリ 3 0 0 に保持される。図 2 0 に示すように、管理テーブル 2 0 0 1 には、領域 R を識別するための R 番号、開始位置、終了位置、領域内に含まれる基本ブロック、属性（精度モード）といった項目が設けられ、領域 R ごとに登録される。

【0 0 6 1】

次に、精度領域解析部 1 3 0 は、順序 0 にしたがって基本ブロック中の未処理のコードをスキャンする（ステップ 1 7 0 3、1 7 0 4、1 7 0 5）。

そして、現在の精度 P よりも低い精度（精度 P が倍精度である場合に単精度）の計算を行うコードが見つかったならば、現在位置（当該コードの前まで）を領域 R の終了位置とする。そして、現在位置（当該コード）を新たに領域 R の開始位置として（ステップ 1 7 0 6、1 7 0 7）、ステップ 1 7 0 3 に戻る。

【0 0 6 2】

一方、現在の精度 P よりも高い精度（精度 P が単精度である場合に倍精度）の

計算を行うコードが見つかったならば、上述した方法1による領域Rの入りと出口とで精度切り替えを行う場合の処理コスト C_s と、方法2による領域Rに含まれる精度Pの計算のそれぞれに対して必要な丸めの処理コスト C_r とを比較する（ステップ1708、1709）。そして、処理コスト C_s の方が大きい場合は、この領域Rを高精度（倍精度）モードで実行するとして、管理テーブル2001に登録する。また、処理コスト C_r の方が大きい場合は、この領域Rを低精度（単精度）モードで実行するとして、管理テーブル2001に登録する。そして、領域R内の基本ブロックの切れ目から出ている制御フローの宛先に、決定された精度モードを伝搬させる（ステップ1710）。

この後、精度領域解析部130は、検出されたコードを次の領域Rの開始位置とし、当該コードの精度を当該次の領域Rの精度Pとして（ステップ1711）、ステップ1703に戻る。

【0063】

基本ブロック中のコードをスキャンして、基本ブロックの切れ目（最後のコード）に到達したならば、精度領域解析部130は、当該基本ブロックがループのバックエッジ、すなわち当該基本ブロックからループの先頭へ戻る制御フローを持つかどうかを調べる（図18、ステップ1712）。

当該基本ブロックがループのバックエッジを持つならば、このループの先頭は処理中の基本ブロックよりも先に処理されていることが順序Oにより保証されている。このループ先頭が属する既に処理された領域R'を、仮に領域Rに含める（ステップ1713）。当該基本ブロックがループのバックエッジを持たないならば、次に精度領域解析部130は、当該基本ブロックの切れ目から他の基本ブロックへの制御フローがあるかどうかを調べる（ステップ1714）。

例えば、図19に示した例において、深さ優先で基本ブロック1、基本ブロック2、基本ブロック4、基本ブロック3の順でコードのスキャンが行われると、基本ブロック4から基本ブロック3への切れ目には制御フローがなく、その他の基本ブロックの切れ目には制御フローがある。

他の基本ブロックへの制御フローがある場合は、ステップ1703に戻り、制御フローの宛先の基本ブロックへ移行し、コードのスキャンを継続する。

【 0 0 6 4 】

一方、他の基本ブロックへの制御フローがない場合、及び基本ブロックがループのバックエッジを持つ場合は、次に、上述した方法 1 による領域 R の入り口と出口とで精度切り替えを行う場合の処理コスト C_s と、方法 2 による領域 R に含まれる精度 P の計算のそれぞれに対して必要な丸めの処理コスト C_r とを比較する（ステップ 1 7 1 5）。そして、処理コスト C_s の方が大きい場合は、この領域 R を高精度（倍精度）モードで実行するとして、管理テーブル 2 0 0 1 に登録する。また、処理コスト C_r の方が大きい場合は、この領域 R を低精度（単精度）モードで実行するとして、管理テーブル 2 0 0 1 に登録する。そして、領域 R 内の基本ブロックの切れ目から出ている制御フローの宛先に、決定された精度モードを伝搬させる（ステップ 1 7 1 6）。

この後、精度領域解析部 1 3 0 は、伝搬された精度モードを現在の精度とし、スキヤンの開始位置を次の基本ブロックの先頭として（ステップ 1 7 1 7）、ステップ 1 7 0 3 に戻る。

【 0 0 6 5 】

精度領域解析部 1 3 0 は、コンパイル対象であるメソッドの全てのコードに対して以上の処理を繰り返し、未処理のコードがなくなったならば、処理を終了する（図 1 7、ステップ 1 7 0 4）

コード生成部 1 2 0 は、精度領域解析部 1 3 0 にて生成されメモリ 3 0 0 に保持されている管理テーブル 2 0 0 1 を参照し、領域 R ごとに精度モードを選択しながら機械語コードを生成する。この結果として、単精度モードを選択された領域 R が上述した単精度領域であり、その他の領域に対しては倍精度モードが選択されることとなる。

【 0 0 6 6 】

以上のように、本実施の形態では、単精度の演算と倍精度の演算とが混在するメソッドに対して、コード解析を行ってメソッド中のコードを単精度領域とそれ以外の領域とに分け、倍精度モードが必要なコード領域を孤立させた上で、各々最適な精度モードを基準とするコード生成を行うため、例えば第 1 の実施の形態と共に本実施の形態を用いれば、倍精度で演算する領域を孤立させ、単精度領域

を単精度モードで扱えるようにすることにより、単精度用の特殊化されたコードの生成対象となるメソッドが多くなる。また、第2の実施の形態と共に本実施の形態を用いることにより、メソッド単位よりもさらに細かいレベルでプログラムの部分ごとに最適なコード生成を行うことが可能となる。

【0067】

上述した各実施の形態は、浮動小数点演算の演算精度として倍精度と単精度とが用いられている場合について説明した。この他、浮動小数点演算の演算精度として拡張精度が用いられる場合がある。この場合においても、上述した各実施の形態をこの拡張精度にまで適用することができる。例えば、第1の実施の形態においては、基準となる精度モード（例えば倍精度モード）に対して、実際にプログラムを走らせて得られる実績（実行プロファイル）に基づいて、拡張精度モードに特殊化された機械語コードを生成することができる。また、第2の実施の形態においては、コールサイトが静的バインドされることを条件に、コンパイルの対象であるメソッド単体で最適化するように、必要ならば拡張精度モードでコード生成を行う。さらに、第4の実施の形態においては、メソッドに対するコード解析によって、拡張精度が連続するコード領域を孤立させ、適切なコード生成を行うことが可能である。

【0068】

【発明の効果】

以上説明したように、本発明によれば、浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、冗長な処理によるオーバーヘッドの発生を回避することが可能となる。

【図面の簡単な説明】

【図1】 第1の実施の形態による浮動小数点演算の精度保証方法が実現されるコンピュータシステムの構成を説明する図である。

【図2】 第1の実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、初期状態におけるデータ構造を示す図である。

【図3】 第1の実施の形態による操作を実現するための、ランタイムルー

チンにより生成されるデータ構造を示す図であり、最初のコンパイル時の典型的なデータ構造を示す図である。

【図 4】 第 1 の実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、特殊化が実行された場合のデータ構造を示す図である。

【図 5】 従来のランタイムルーチンにより生成されるデータ構造を示す図である。

【図 6】 第 1 の実施の形態によるプログラムの特殊化の行程を示すフローチャートである。

【図 7】 第 2 の実施の形態において、コンパイル対象メソッドの呼び出し側が既にコンパイルされている場合のコード生成を説明する図である。

【図 8】 第 2 の実施の形態において、コンパイル対象メソッドの呼び出し側が未だコンパイルされていない場合のコード生成を説明する図である。

【図 9】 第 2 の実施の形態によるコンパイルの手順を説明するフローチャートである。

【図 1 0】 第 3 の実施の形態におけるコンパイラの機能を説明する図である。

【図 1 1】 第 3 の実施の形態にて用いられる過剰精度計算表を示す図である。

【図 1 2】 過剰精度最適化部が所定の計算に関して過剰精度か否かを判定する処理の流れを説明するフローチャートである。

【図 1 3】 過剰精度計算と判定された計算を低精度の演算に置き換える処理の流れを説明するフローチャートである。

【図 1 4】 第 3 の実施の形態にて具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図であり、単精度浮動小数点値に値 0.5 を乗算する倍精度計算を単精度化する例を疑似コードで示す図である。

【図 1 5】 第 3 の実施の形態にて具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図であり、単精度浮動

小数点値の平方根 $\text{sqrt}(v)$ を倍精度で求めて単精度に丸める計算を単精度化する例を疑似コードで示す図である。

【図 1 6】 第 4 の実施の形態におけるコンパイラの機能を説明する図である。

【図 1 7】 第 4 の実施の形態において、精度領域解析部が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

【図 1 8】 第 4 の実施の形態において、精度領域解析部が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

【図 1 9】 所定のソースコードにおける基本ブロックを示す図である。

【図 2 0】 第 4 の実施の形態にて用いられる管理テーブルの例を示す図である。

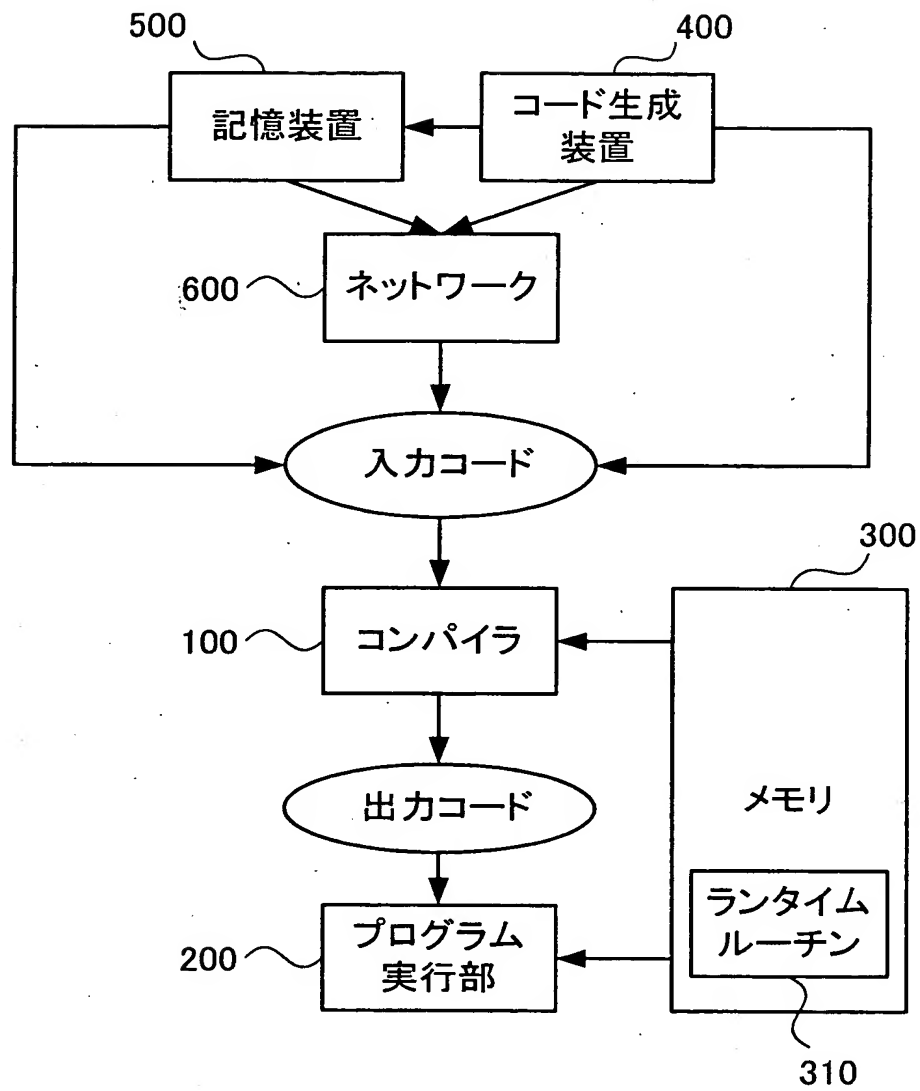
【符号の説明】

1 0 0 … コンパイラ、 1 1 0 … 過剰精度最適化部、 1 2 0 … コード生成部、 1 3 0 … 精度領域解析部、 2 0 0 … プログラム実行部、 3 0 0 … メモリ、 3 1 0 … ランタイムルーチン、 3 2 0 … 過剰精度計算表、 4 0 0 … コード生成装置、 5 0 0 … 記憶装置、 6 0 0 … ネットワーク

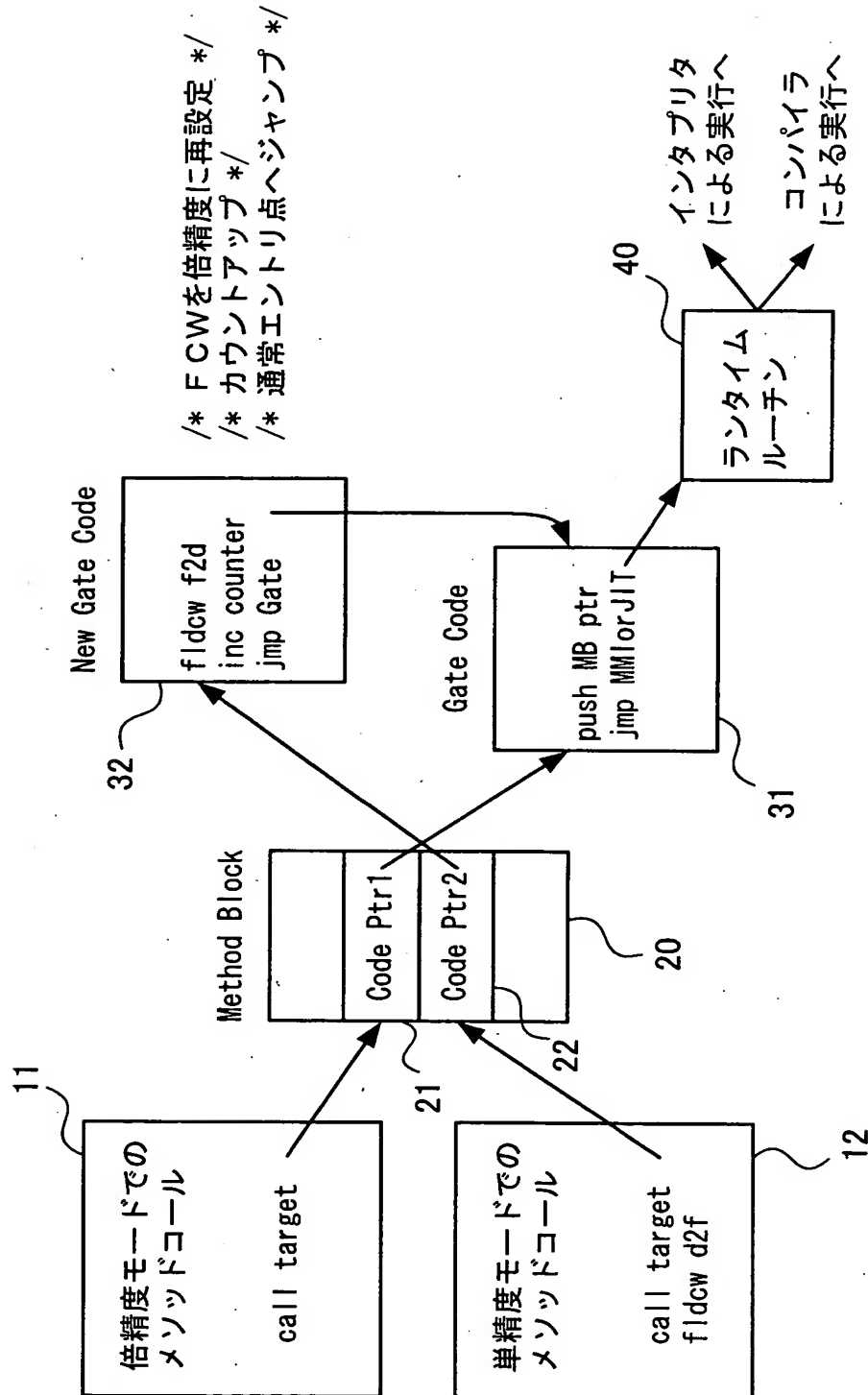
【書類名】

図面

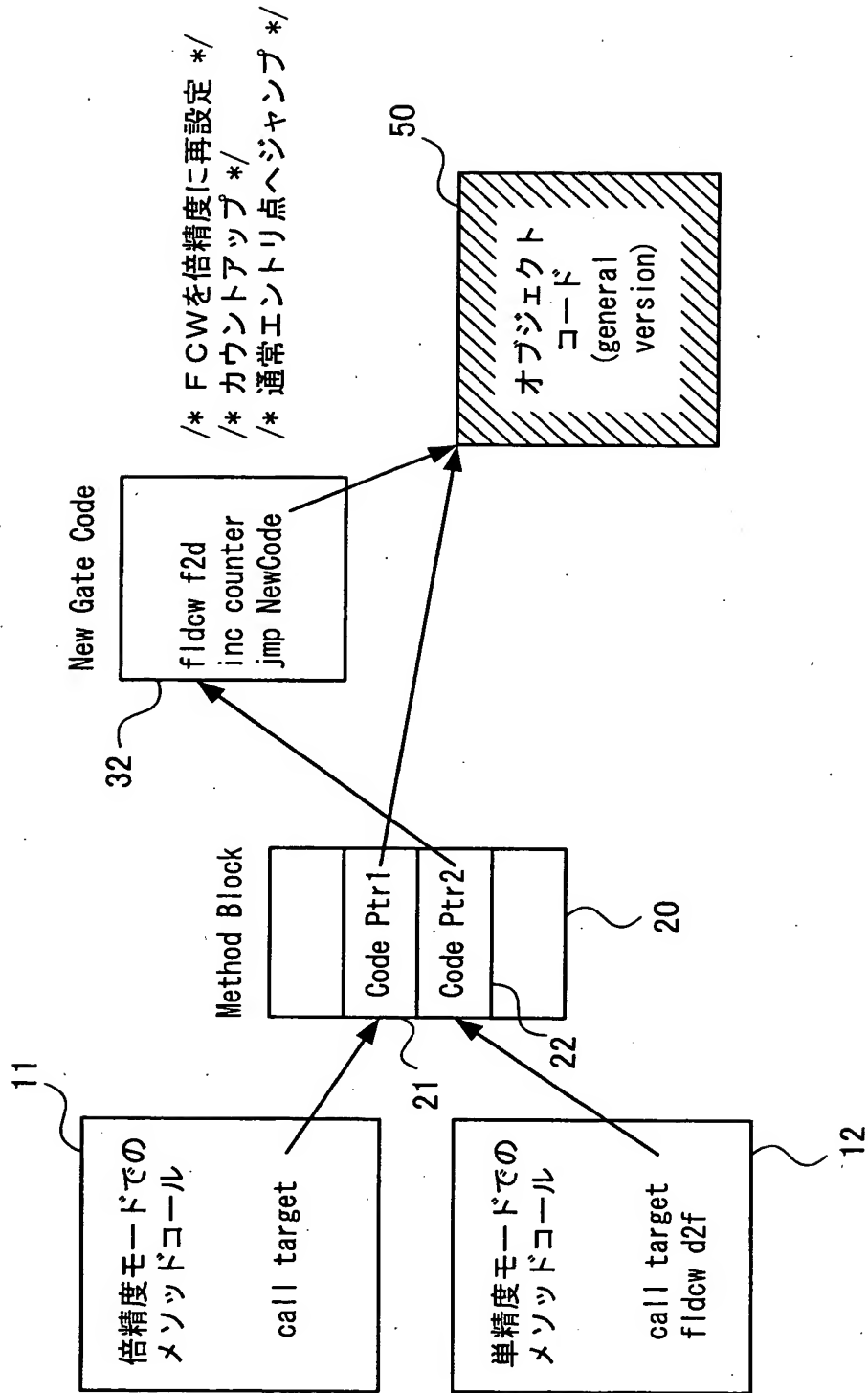
【図 1】



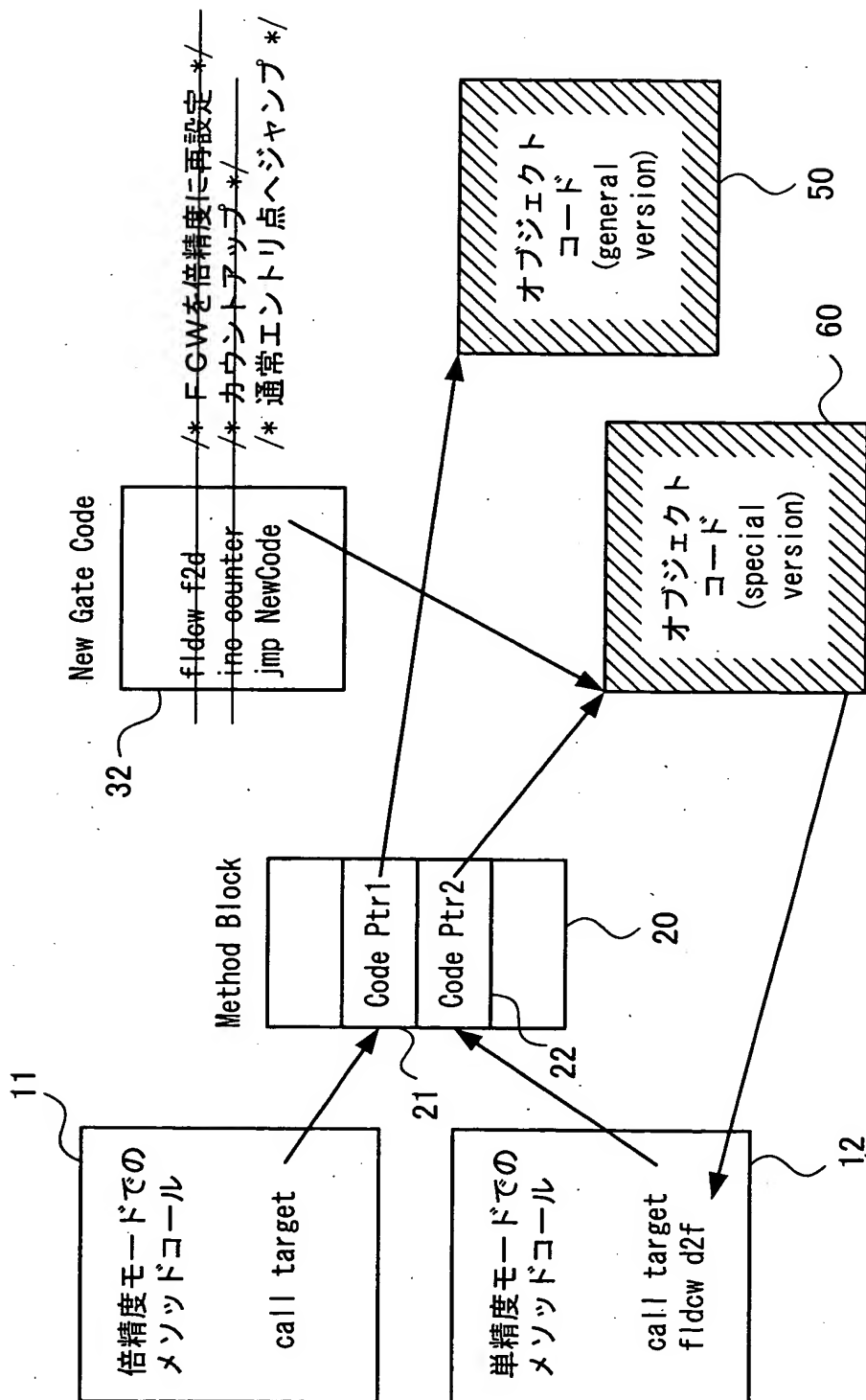
【図 2】



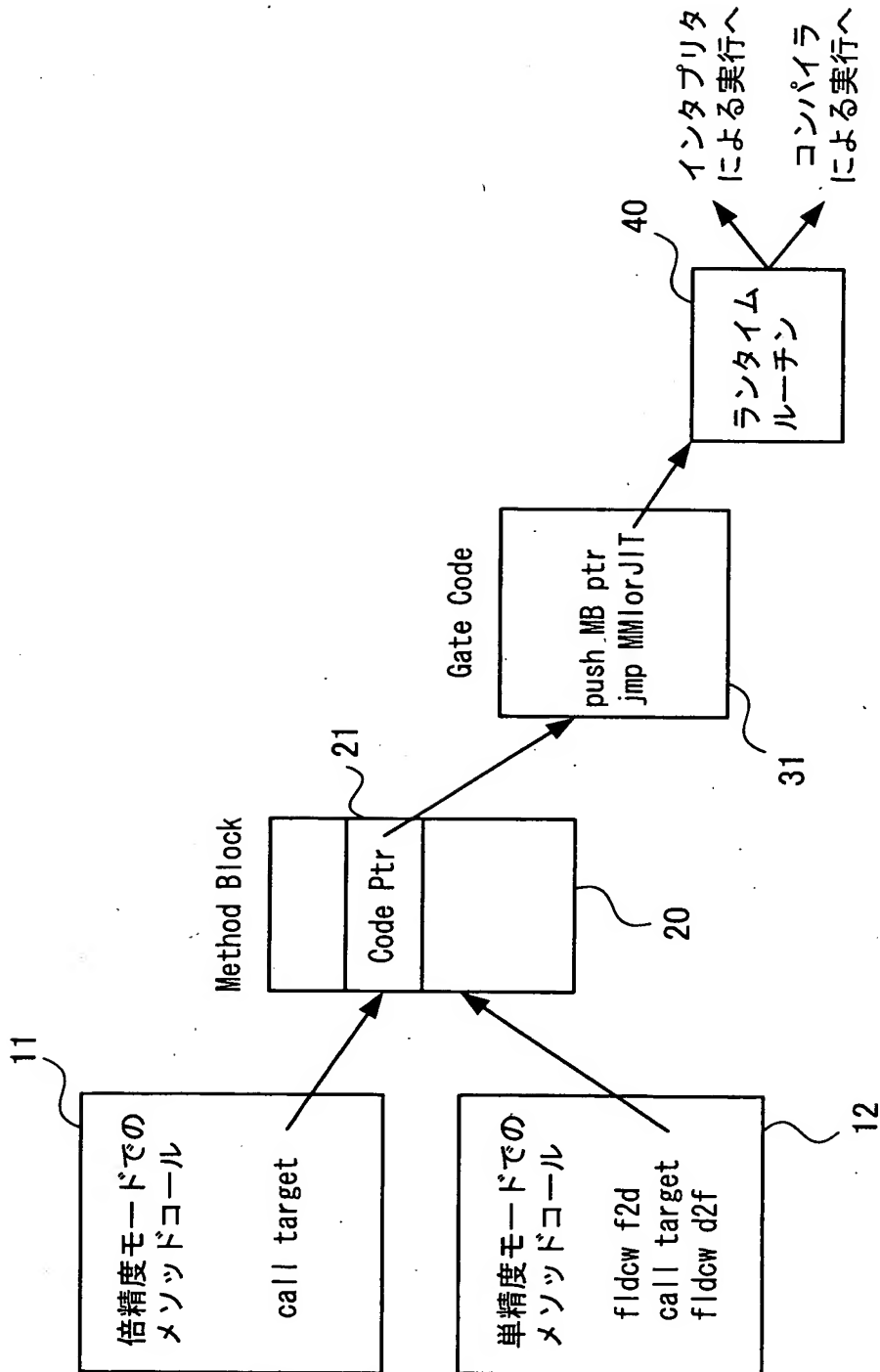
【図 3】



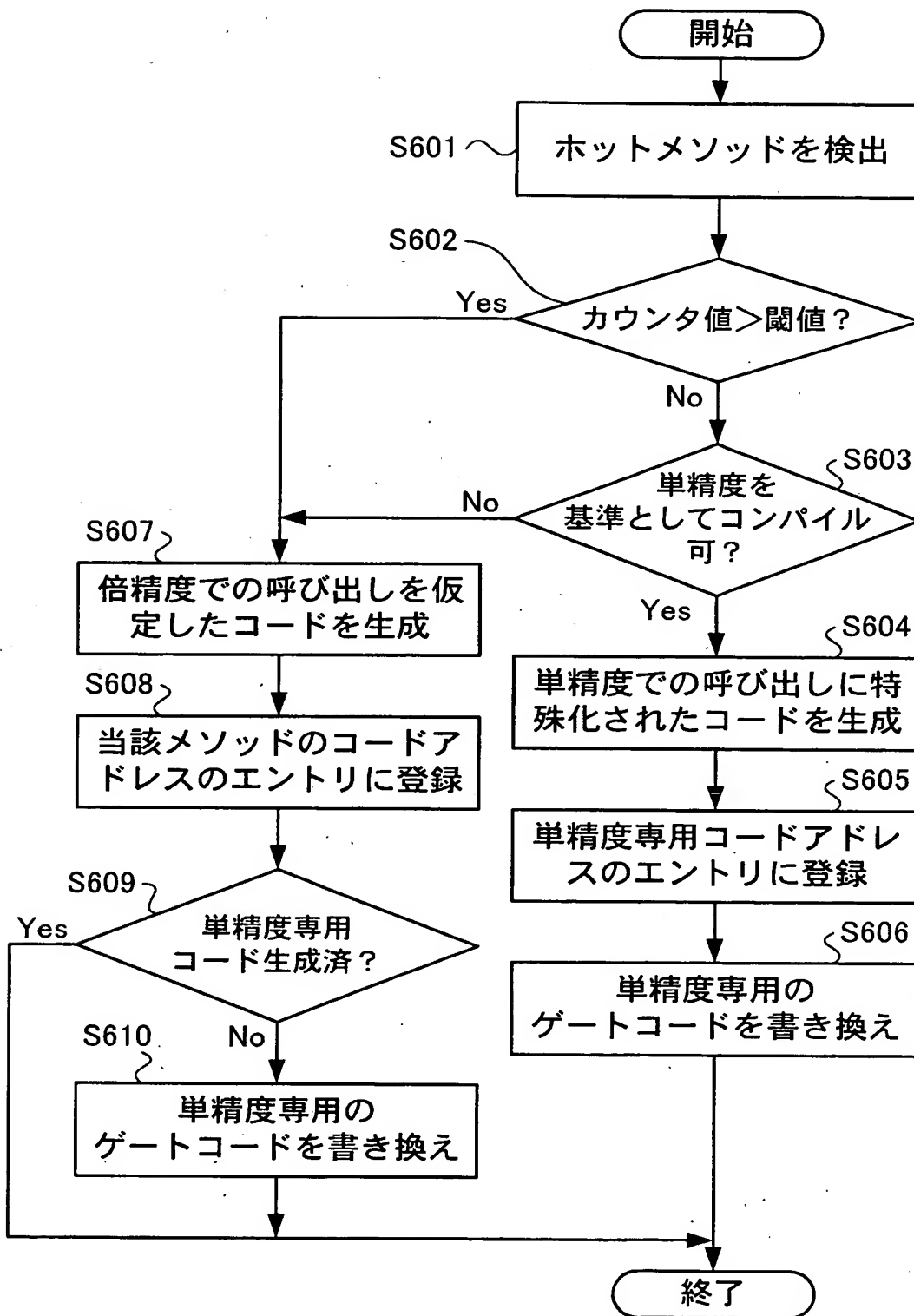
【図 4】



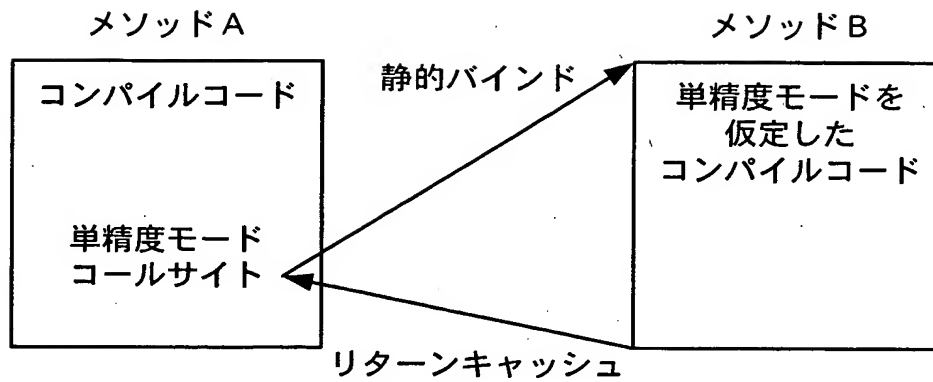
【図 5】



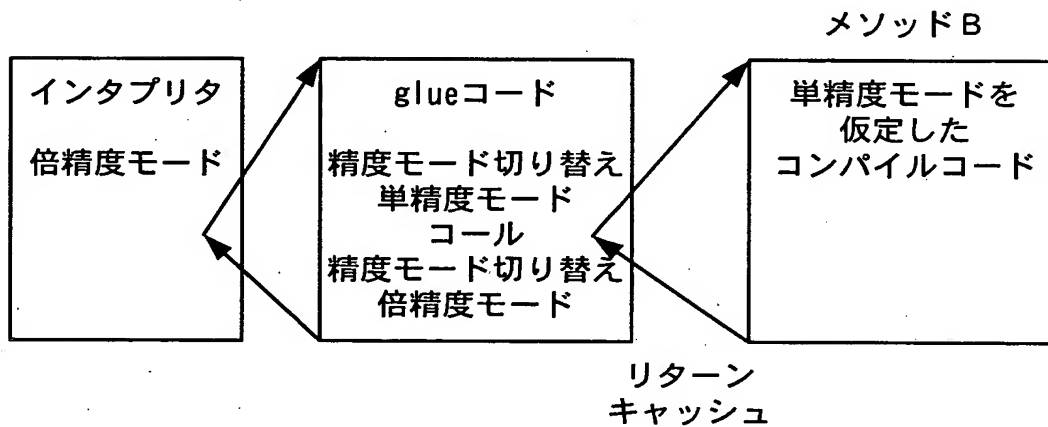
【図 6】



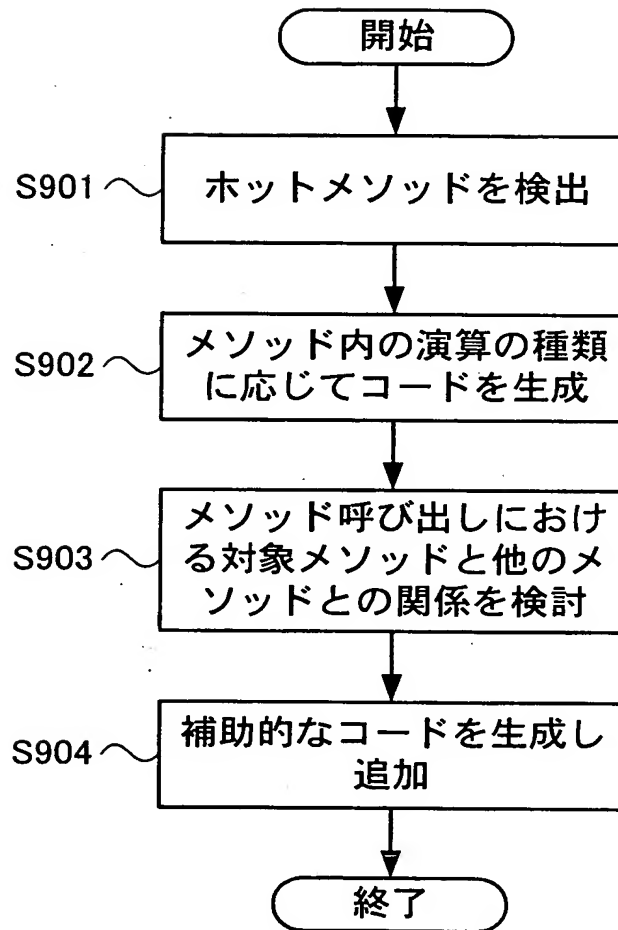
【図 7】



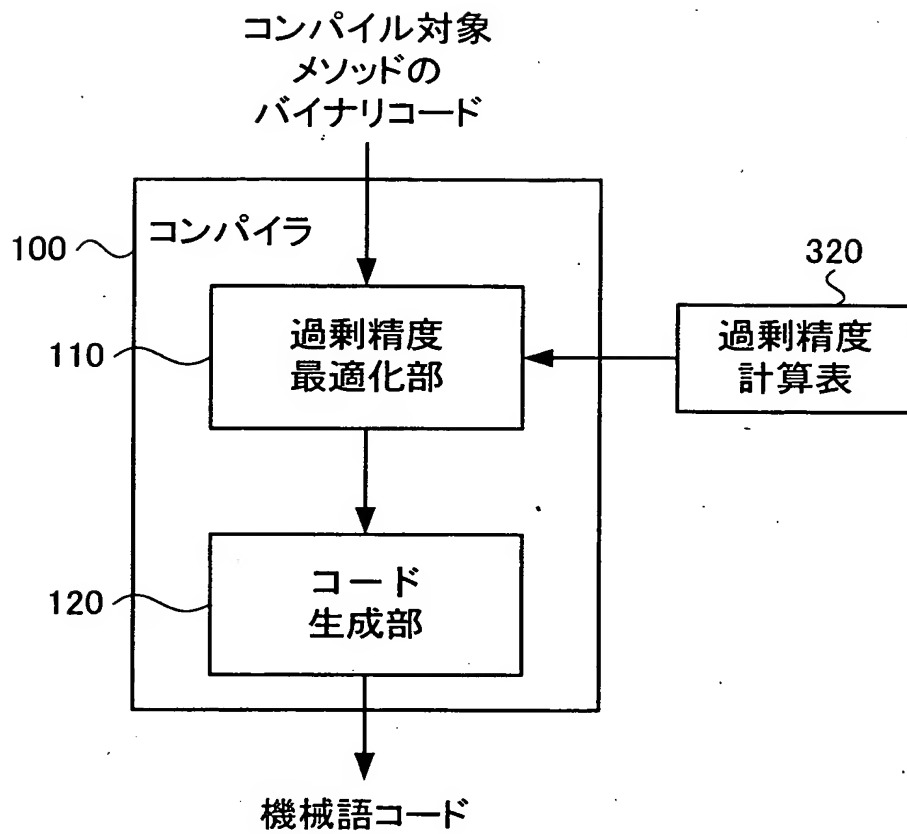
【図 8】



【図 9】



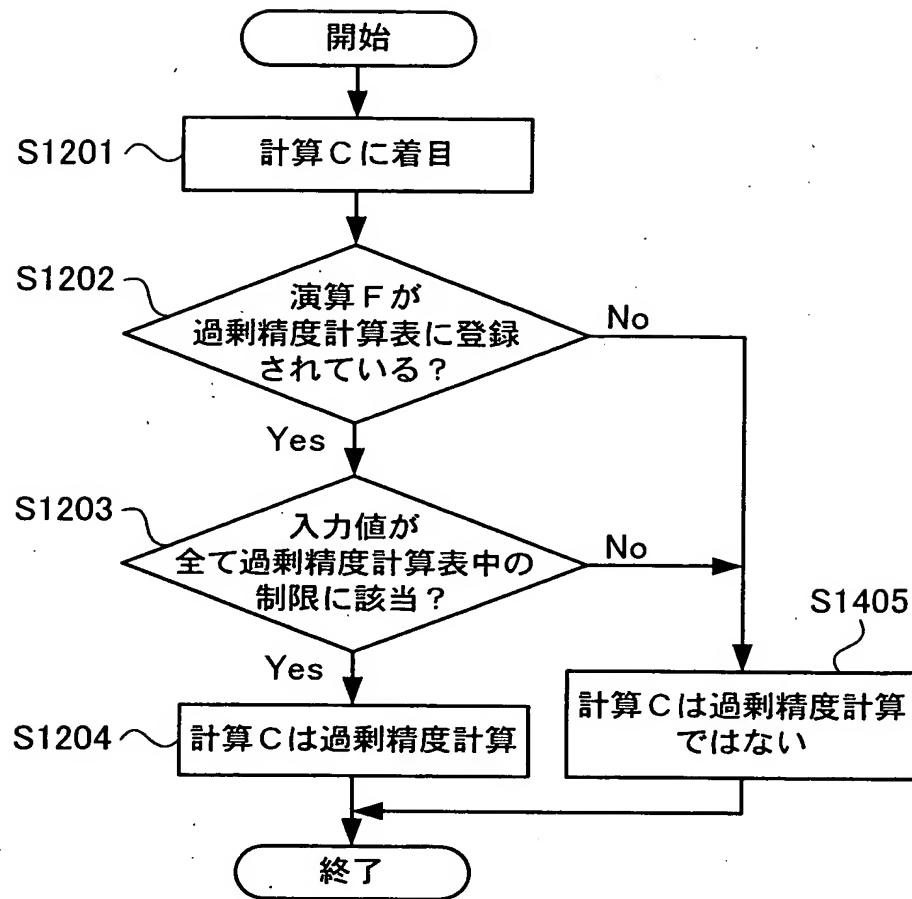
【図10】



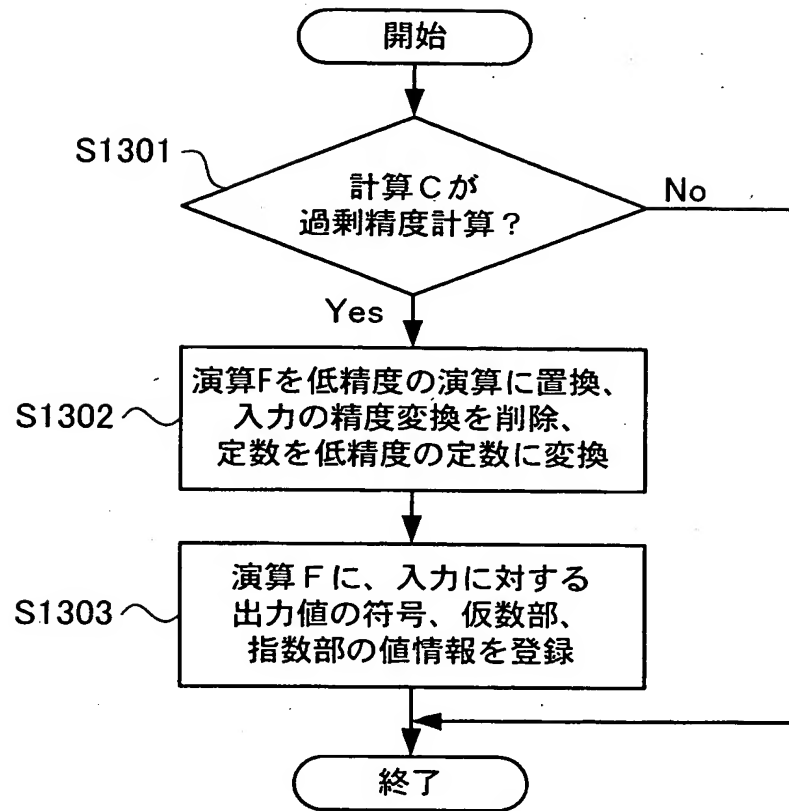
【図11】

計算種別	入力1の制限	入力2の制限	...
DMUL	仮数部が単精度	仮数部が1	
DSQRT+D2F	仮数部が単精度	N.A.	
・ ・ ・			

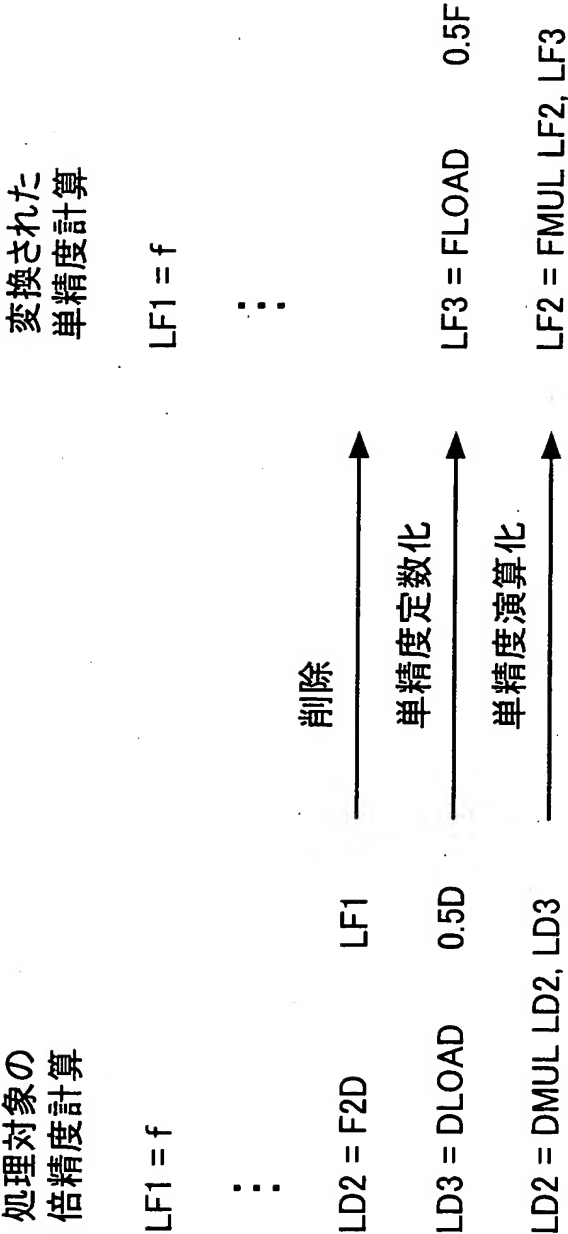
【図 12】



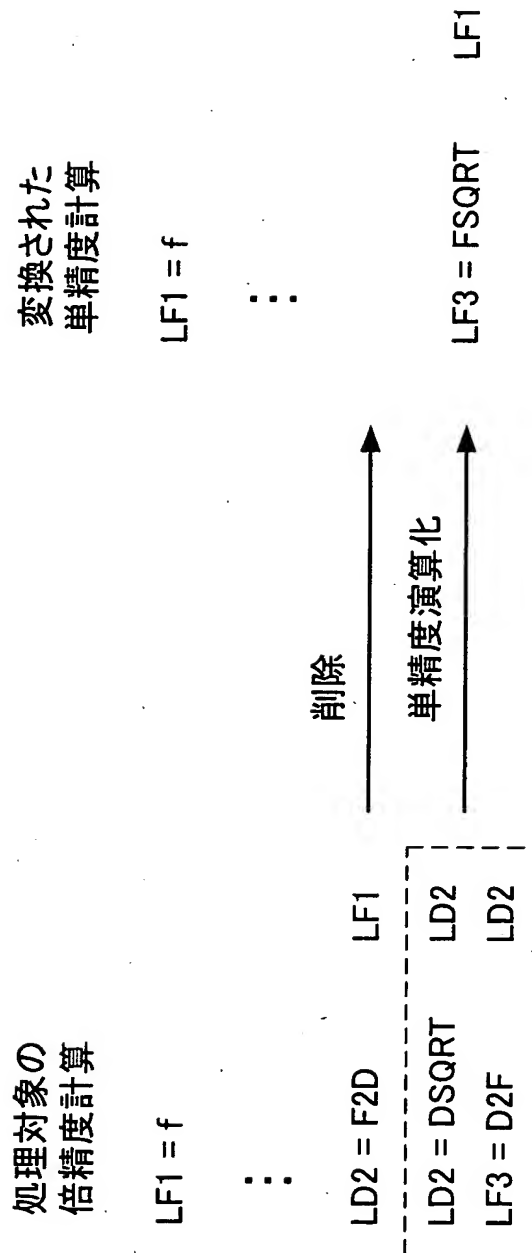
【図 13】



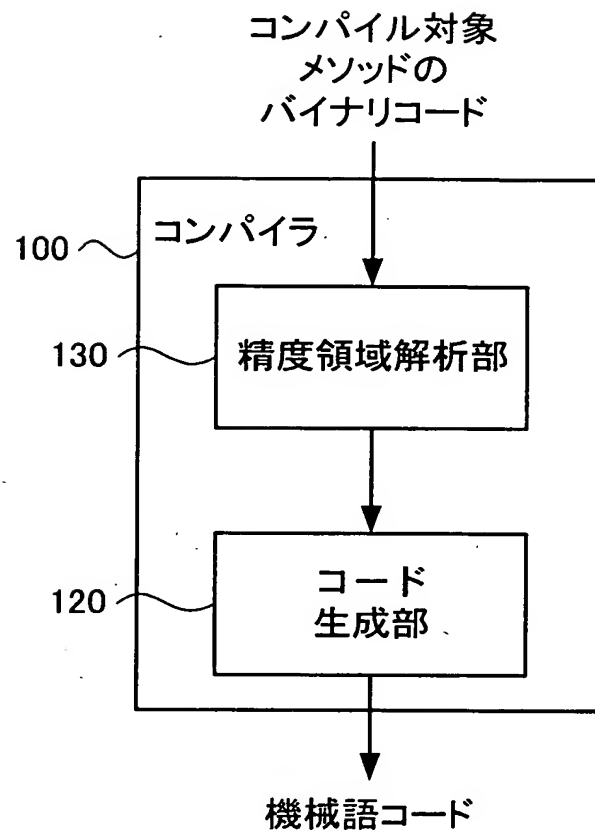
【図 1 4】



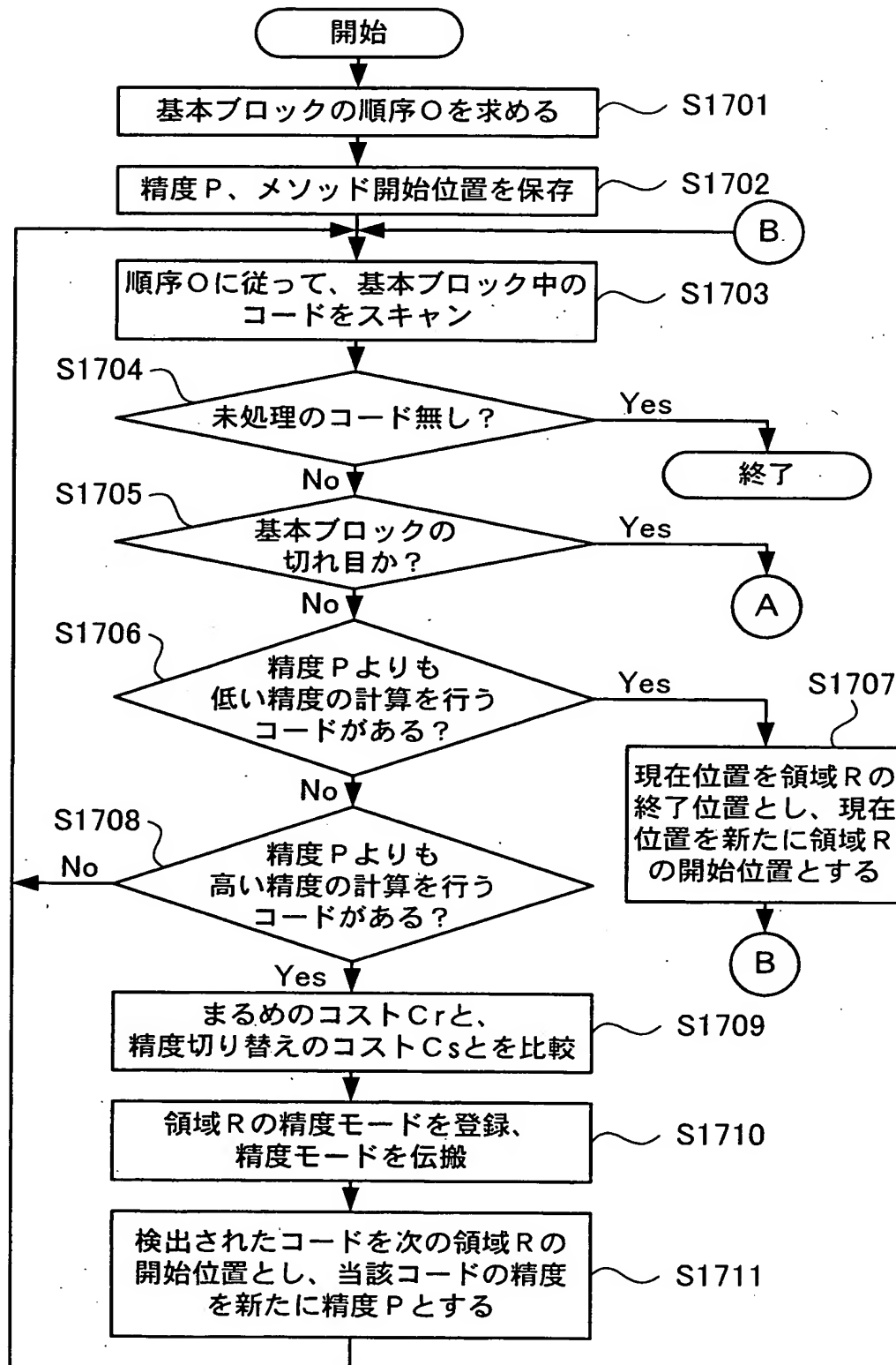
【図 15】



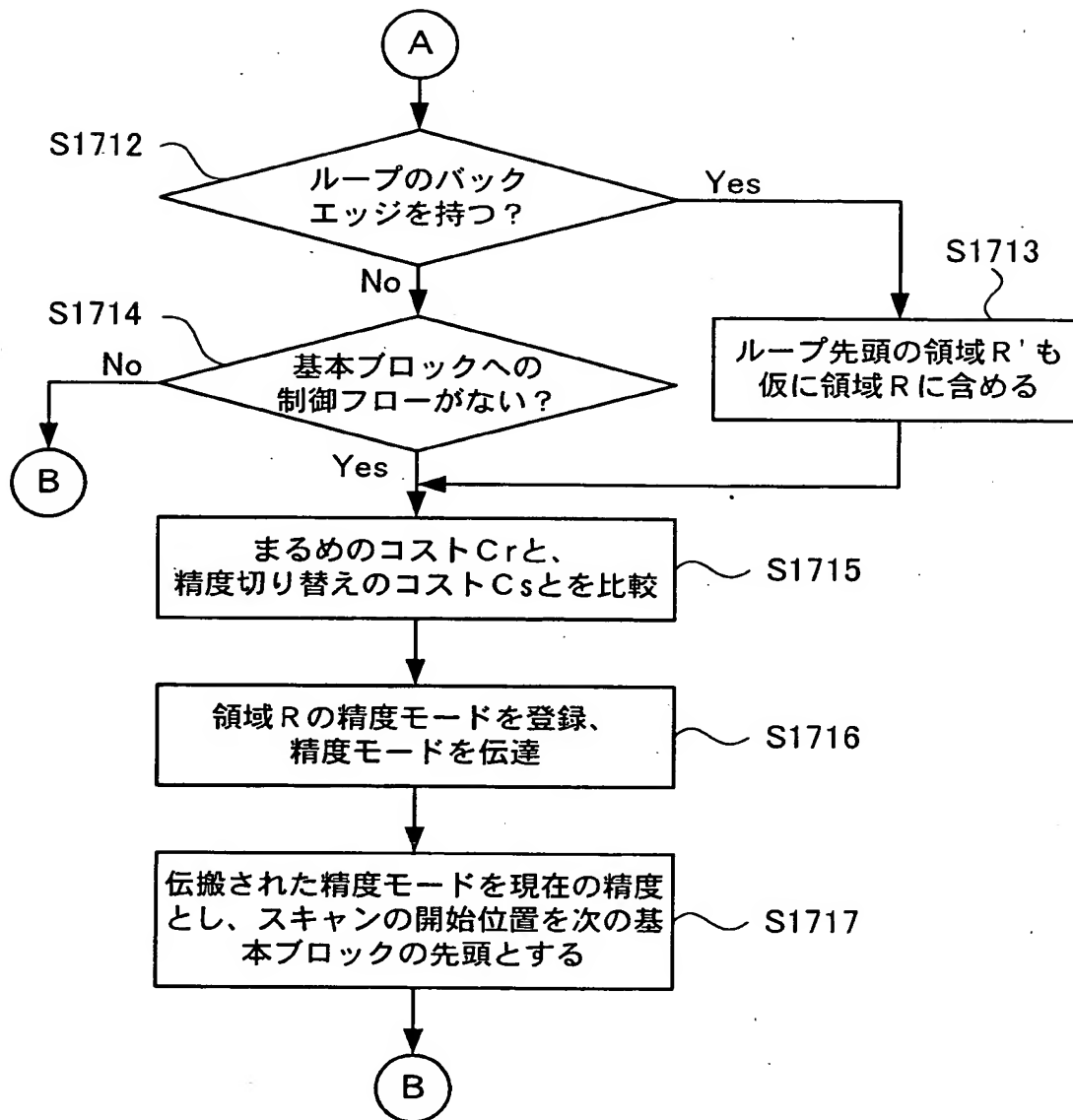
【図 1 6】



【図 1,7】



【図18】



【図 19】

(A)

```

if (cond) {
    statement1
    statement2
} else {
    statement21
    statement22
}
statement3
    
```

(B)

```

基本ブロック 1
    test cond
    if true goto 基本ブロック 2
    goto 基本ブロック 3

基本ブロック 2
    statement11
    statement12
    goto 基本ブロック 4

基本ブロック 3
    statement21
    statement22
    goto 基本ブロック 4

基本ブロック 4
    statement3
    
```

【図 2 0】

2001

R 番 号	開 始 位 置	終 了 位 置	領 域 に 含 ま れ る 基 本 ブ ロ ッ ク	属 性
1	基 本 ブ ロ ッ ク 1 内 3 番 目 の 命 令	基 本 ブ ロ ッ ク 3 内 2 番 目 の 命 令	1, 2, 3	単 精 度
2	⋮	⋮	⋮	⋮
⋮				

【書類名】 要約書

【要約】

【課題】 浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、オーバーヘッドや解析処理の無駄の発生を回避する。

【解決手段】 コンピュータに搭載されたコンパイラ 1 0 0 において、浮動小数点演算における倍精度モードでメソッドのオブジェクトコードを生成すると共に、このメソッドが単精度モードで頻繁に呼び出され、かつ単精度モードで実行することにより処理コストを削減できる場合に、単精度モードでメソッドの第 2 のコードを生成する。あるいは、コンパイルしようとする対象メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらから呼ばれるかに関わらず、この対象メソッド内の演算の種類に応じて演算精度を設定してオブジェクトコードを生成し、対象コードとこの対象コードを呼び出す他のコードまたはこの対象コードに呼び出される他のコードとの関係に基づいて精度モードを合わせるためのコードを生成し追加する。

【選択図】 図 1

認定・付加情報

特許出願の番号	特願 2002-224207
受付番号	50201136193
書類名	特許願
担当官	佐々木 吉正 2424
作成日	平成14年 9月 9日

<認定情報・付加情報>

【特許出願人】

【識別番号】	390009531
【住所又は居所】	アメリカ合衆国10504、ニューヨーク州 アーモンク ニュー オーチャード ロード
【氏名又は名称】	インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】	100086243
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	坂口 博

【代理人】

【識別番号】	100091568
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	市位 嘉宏

【代理人】

【識別番号】	100108501
【住所又は居所】	神奈川県大和市下鶴間1623番14 日本アイ・ビー・エム株式会社 知的所有権
【氏名又は名称】	上野 剛史

【復代理人】

【識別番号】	100104880
【住所又は居所】	東京都港区赤坂5-4-11 山口建設第2ビル 6F セリオ国際特許事務所
【氏名又は名称】	古部 次郎

出 願 人 履 歴 情 報

識別番号 [390009531]

1. 変更年月日 2002年 6月 3日

[変更理由] 住所変更

住 所 アメリカ合衆国10504、ニューヨーク州 アーモンク ニ
ュー オーチャード ロード

氏 名 インターナショナル・ビジネス・マシーンズ・コーポレーショ
ン